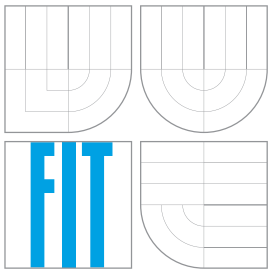
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

MAPOVÁNÍ VYHLEDÁVACÍCH TABULEK Z JAZYKA P4 DO TECHNOLOGIE FPGA

MAPPING OF MATCH TABLES FROM P4 LANGUAGE TO FPGA TECHNOLOGY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL KEKELY

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN KOŘENEK, Ph.D.

BRNO 2016

Zadání diplomové práce/18100/2015/xkekel01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Kekely Michal, Bc.**
Obor: Inteligentní systémy
Téma: **Mapování vyhledávacích tabulek z jazyka P4 do technologie FPGA**
Mapping of Match Tables from P4 Language to FPGA Technology
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s jazykem P4, zejména s vyhledávacími tabulkami MAT (Match Action Tables).
2. Nastudujte architekturu FPGA Virtex 7.
3. Navrhněte vhodnou hardwarovou architekturu pro mapování vyhledávacích tabulek MAT do FPGA.
4. Navrženou architekturu implementujte.
5. Ověřte parametry vytvořené implementace. Zaměřte se zejména na rychlost vyhledávání a množství využitých zdrojů.
6. V závěru diskutujte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kořenek Jan, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 25. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Diplomová práca sa zaoberá návrhom a implementáciou mapovania vyhľadávacích tabuliek jazyka P4 do technológie FPGA. Cieľom práce bolo popísať kľúčové princípy, ktoré je potrebné pochopiť na vytvorenie návrhu samotného mapovania a fungovania potrebných algoritmov, tieto princípy aplikovať v rámci implementácie a analyzovať výsledné riešenie z pohľadu rýchlosti a náročnosti na pamäť a zdroje cieľovej architektúry. Výsledok práce poskytuje konfigurovateľnú hardvérovu jednotku schopnú klasifikovať pakety a jej prepojenie na vyhľadávacie tabuľky jazyka P4. Riešenie využíva algoritmus DCFL a oproti algoritmom HiCuts a HyperCuts dosahuje v najhoršom prípade porovnateľné priepustnosti, ale vyžaduje podstatne menej pamäte.

Abstract

This thesis deals with design and implementation of mapping of match action tables from P4 language to FPGA technology. Goal of the thesis was to describe key principles, which need to be understood in order to design such a mapping and function of algorithms needed, apply these principles by implementing them and analyze the speed and memory requirements of such an implementation. Outcome provides configurable hardware unit capable of classifying packets and connection between the unit and match action tables from P4 language. The implementation is based on DCFL algorithm and requires less memory compared to HiCuts and HyperCuts algorithms while being comparably fast at worst-case scenarios.

Kľúčové slová

P4, klasifikácia, vyhľadávacia tabuľka, kukučie hašovanie, trie, FPGA, filtrovanie paketov, DCFL

Keywords

P4, classification, hash table, cuckoo hashing, trie, FPGA, packet filtering, DCFL

Citácia

KEKELY, Michal. *Mapování vyhledávacích tabulek z jazyka P4 do technologie FPGA*. Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Kořenek Jan.

Mapování vyhledávacích tabulek z jazyka P4 do technologie FPGA

Prehlásenie

Prehlasujem, že túto diplomovú prácu som vypracoval samostatne pod vedením pána Ing. Jana Kořenka, Ph.D., a uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Michal Kekely
22. mája 2016

Poďakovanie

Chcel by som poďakovať vedúcemu práce Ing. Janovi Kořenkovi, Ph.D., za ochotu a poskytnutie cenných rád a odbornej pomoci pri tvorbe tejto práce.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam zdrojových kódov

Kapitola 1

Úvod

Znakom dnešnej doby je stále rýchlejší vývoj vo všetkých oblastiach ľudskej činnosti. Tento trend je možné pozorovať aj v oblasti výpočtovej techniky, ktorá patrí medzi najrýchlejšie rastúce. Počítačové siete nie sú žiadnou výnimkou. Počet používateľov rastie, používatelia používajú počítačové siete častejšie. Stále viac a viac rozširujú aj dátovo náročnejšie sieťové komunikácie ako je napríklad streaming videí, hudby a sťahovanie digitálnych kópií rôznych iných médií. Navyše sa mení aj samotný charakter sietí. Koncové zariadenia sa stávajú mobilnejšími, čo vedie k dynamickejším sieťam.

S rastom počítačových sietí narastá aj počet bezpečnostných incidentov. Na odhalenie a predchádzanie týmto incidentom je potrebné, aby prvky sieťovej infraštruktúry slúžiace na monitorovanie a analýzu siete nezaostávali svojou výkonnosťou za ostatnými sieťovými zariadeniami. Výpočtový výkon, ktorý monitorovacie zariadenia vyžadujú, však často prevyšuje možnosti konvenčných výpočtových prvkov. Klasický prístup riešenia problému softvérovo nám poskytuje vysokú flexibilitu a jednoduchosť, avšak na úkor výkonu a efektivity. Alternatívou je využitie hardvérovej akcelerácie, kedy niektoré kritické časti výpočtu prenesieme do špeciálne navrhnutých hardvérových jednotiek. Tieto hardvérové jednotky nám poskytujú možnosť výpočet paralelizovať, využívať zreťazené spracovanie a hardvérové jednotky môžeme navrhnúť a optimalizovať s ohľadom na funkcionalitu, ktorú majú zabezpečovať. Takto sme schopní dosiahnuť až niekoľkonásobné zrýchlenie, úsporu výpočtových zdrojov a spotreby energie.

Hardvérové jednotky však so sebou prinášajú zásadné problémy. Hlavným problémom je úzka špecializácia hardvérových zariadení, čo vedie k zníženiu flexibility riešenia. Navyše vytváranie robustných hardvérových jednotiek je často oveľa zložitejšie ako vytvorenie ekvivalentných softvérových jednotiek.

Meniaci sa charakter sietí, ich použitia a využívanie hardvérovej akcelerácie má navyše za následok, že niektoré používané sieťové architektúry prestávajú byť vhodnými. Najmä staršie sieťové architektúry neboli navrhované s ohľadom na komplexnosť dnešných sietí a neposkytujú dostatočnú škálovateľnosť, flexibilitu, rýchlosť či úroveň dosiahnutej abstrakcie. Jedným z možných riešení tohoto problému je rozdelenie funkcionality medzi jednoduchý výkonný hardvér a sofistikovaný softvér. Na tejto myšlienke je založený aj jazyk P4 (predstavený v článku [1]). Tento novovznikajúci jazyk má za cieľ poskytnúť možnosť urýchliť spracovanie paketov využitím rôznych hardvérových a softvérových riešení. Navyše programátor môže pracovať s vysokou úrovňou abstrakcie a vytvorené riešenie je dostatočne flexibilné a jednoducho rozšíriteľné.

Jednu z najdôležitejších častí jazyka P4 tvoria vyhľadávacie tabuľky (anglicky match action tables), ktoré riešia všeobecný problém klasifikácie paketov. Klasifikácia paketov

(popísaná napríklad v článku [6]) predstavuje proces, kedy sa snažíme prichádzajúci paket (na základe hodnôt z jeho hlavičiek) zaradiť do určitej triedy, určenej pravidlom z množiny daných pravidiel. Klasifikácia paketov je dôležitá najmä z pohľadu bezpečnosti sietí, kde sa využíva na rozpoznanie, ktoré pakety patria ku ktorému logickému spojeniu a prípadne, ktoré pakety sú nežiadúce a mali by byť blokované. Už pri rýchlosti 10Gbps je však v najhoršom prípade nutné klasifikovať vyše 31 miliónov paketov, čo môže byť pre softvérové riešenie nezvládnuteľné. Preto klasifikácia paketov a teda aj vyhľadávacie tabuľky jazyka P4 predstavujú vhodného kandidáta na využitie hardvérovej akcelerácie.

Cieľom tejto diplomovej práce je najprv rozobrať princípy, na ktorých je založené mapovanie vyhľadávacích tabuliek jazyka P4 do technológie FPGA. Následne tieto princípy aplikovať v podobe návrhu a implementácie konfigurovateľnej hardvérovej jednotky schopnej klasifikácie paketov a nakoniec výsledné riešenie analyzovať hlavne z pohľadu priepustnosti a náročnosti na zdroje cieľovej architektúry. Výsledné riešenie teda poskytuje prepojenie medzi abstraktnejšou sieťovou architektúrou a hadvérovou jednotkou schopnou urýchliť klasifikáciu paketov.

Práca obsahuje sedem kapitol. Kapitola 2 obsahuje popis princípov počítačových sietí a základné protokoly využívané pri sieťovej komunikácii. V kapitole 3 je popísaná nová sieťová architektúra SDN. Na túto kapitolu nadväzuje kapitola 4 obsahujúca popis samotného jazyka P4, ktorý ďalej rozširuje niektoré princípy architektúry SDN. Kapitola 5 popisuje proces klasifikácie paketov a algoritmy, ktoré sa na klasifikáciu paketov využívajú. Kapitola 6 obsahuje popis návrhu a implementácie hardvérovej jednotky a mapovania vyhľadávacích tabuliek jazyka P4 na konfiguráciu tejto jednotky. Dosiahnuté výsledky riešenia sú zhrnuté v kapitole 7. Nakoniec v kapitole 8 je krátke zhrnutie.

Kapitola 2

Počítačové siete

Táto kapitola rozoberá základné princípy počítačových sietí a využívané modely a protokoly. Rozbor začína popisom základného princípu sieťovej komunikácie. V druhej sekcii sú bližšie popísane sieťové modely ISO/OSI a TCP/IP. Nakoniec posledná sekcia obsahuje popis niektorých vybraných sieťových protokolov. Poznatky obsiahnuté v tejto kapitole vychádzajú z knihy [3] a z mojej bakalárskej práce [9].

2.1 Základný princíp

Počítačové siete sú telekomunikačné siete, ktoré umožňujú zariadeniam výmenu dát. Prvky, ktoré sa v týchto sieťach vyskytujú, môžeme rozdeliť na dva druhy. Prvým sú koncové zariadenia, ide o zariadenia, ktoré sú buď odosielateľom správy alebo príjemcom tejto správy. Medzi koncové zariadenia patria osobné počítače, tablety, IP telefóny a mnohé iné.

Správa sa však od odosielateľa k príjemcovi musí nejakým spôsobom dostať. Pokiaľ sú priamo prepojení nenastáva problém. Akonáhle však potrebujeme prepojiť väčšie množstvo zariadení, ktoré môžu byť na rôznych miestach, nebolo by veľmi výhodné prepojiť priamo každé zariadenie s každým. Využívame preto druhú skupinu zariadení, a to zariadenia sprostredkovateľské. Ide o zariadenia, ktoré zapezpečujú smerovanie a doručenie dát do správneho koncového zariadenia. Do tejto kategórie spadajú rozbočovače a smerovače. Tieto zariadenia sú často pre bežného užívateľa skryté a o ich existencii ani nemusí vedieť.

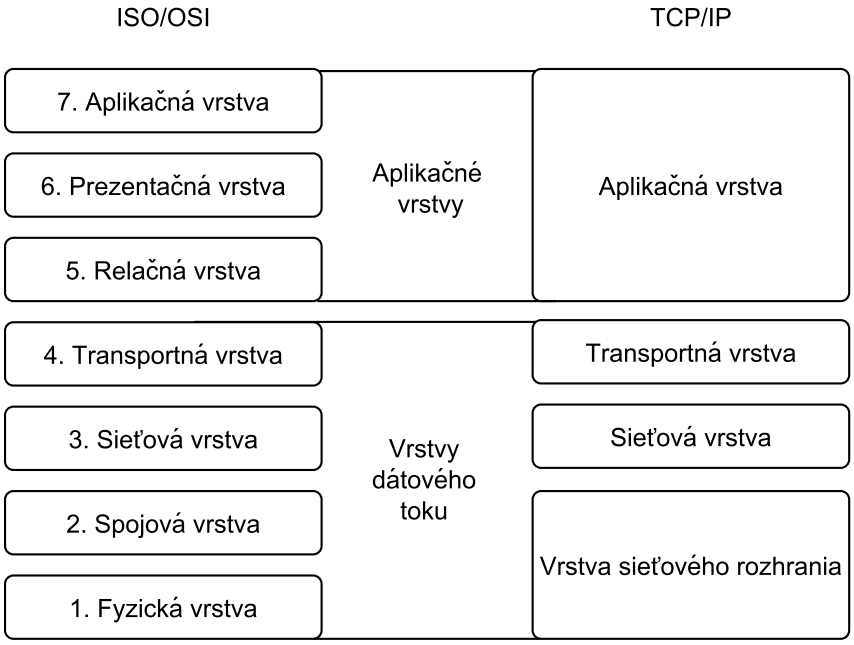
Všetky tieto zariadenia musia byť nejakým spôsobom prepojené. Na to slúži prenosové médium. V počítačových sieťach je týmto médiom medený alebo optický kábel, prípadne bezdrôtová technológia.

Pre úspešnú komunikáciu ďalej potrebujeme, aby si zariadenia rozumeli. Potrebujeme teda dátam a metadátam, ktoré si vymieňajú, priradiť význam. Toto je úlohou sieťových protokolov. Tieto protokoly určujú štruktúru a význam dát. Zároveň môžu poskytovať mechanizmy pre riadenie prenosu dát. Základné protokoly sú definované dvoma modelmi: modelom ISO/OSI a modelom TCP/IP.

2.2 Modely ISO/OSI a TCP/IP

Ide o vrstvovú abstraktnú reprezentáciu vytvorenú ako pomôcku pre návrh sieťových protokolov. Vrstvy je možné vidieť na obrázku 2.1. Model OSI predstavuje formálnejší prístup. Rozdeľuje proces sieťovej komunikácie na 7 logických vrstiev, pričom každej vrstve priraďuje unikátnu funkcionalitu a rozhranie na komunikáciu s ostatnými vrstvami. Informácie

sú postupne propagované medzi vrstvami počínajúc aplikačnou na strane odosielateľa, pokračujúc naprieč všetkými vrstvami až po fyzickú, ďalej cez médium do prijímacej strany, kde je spracovaná vrstvami v opačnom poradí.



Obr. 2.1: Modely ISO/OSI a TCP/IP.

Tento spôsob oddelenia funkcionality výrazne zjednodušuje tvorbu nových protokolov či zariadení a ich začlenenie do existujúcej siete. Pokiaľ napríklad zmeníme prenosové médium, postačí zmeniť protokol najnižšej vrstvy.

V praxi prechod informácie jednotlivými vrstvami znamená pridanie riadiacich informácií danej vrstvy vo forme hlavičky, prípadne päty. Hovoríme o zabaľovaní dát.

V dnešnej dobe sa využíva hlavne TCP/IP model. Tento model má len 4 vrstvy:

1. Vrstva sieťového rozhrania tvorí rozhranie hardvérových zariadení a prenosových médií. Ide o najnižšiu vrstvu modelu, ktorá zabezpečuje hardvérovú nezávislosť celého modelu. Ďalej zabezpečuje prípravu dát na prenos a samotný prenos dát po prenosovom médiu. Protokoly tejto vrstvy sú napríklad Ethernet alebo Token Ring. Dátové entity na tejto vrstve sa nazývajú rámce (anglicky frame).

2. Sieťová vrstva sa stará o zasielanie dát naprieč potenciálne viacerými sieťami. Tento proces sa nazýva smerovanie. Táto vrstva má dve základné funkcie. Adresuje a identifikuje zariadenia v sieti pomocou hierarchického systému IP adries a smeruje dáta zo zdroja k cieľu tým, že ich preposiela vždy na ďalší smerovač, ktorý je bližšie pri cieli. Táto vrstva však poskytuje len nespoľahlivý dátový prenos. Protokolom tejto vrstvy je protokol IP a ďalšie protokoly, ktoré funkčnosť protokolu IP podporujú, ako ICMP, IGMP a smerovacie protokoly EIGRP, OSPF a RIP (tieto protokoly sú často priraďované do akejsi medzivrstvy medzi vrstvou sieťovou a transportnou). Dátové entity tejto vrstvy nazývame pakety (anglicky packet).

3. Transportná vrstva slúži na základnú komunikáciu medzi aplikáciami na koncových zariadeniach. Obsahuje teda mechanizmus pre adresovanie jednotlivých aplikácií bežiacich na jednom zariadení pomocou čísel portov. Ďalej definuje mechanizmy pre kontrolu zahltenia a toku, kontrolu správnosti prijatých dát a segmentáciu. Úlohou tejto vrstvy je prenos správ nezávisle na prenosovom médiu či type siete. Protokoly transportnej vrstvy delíme na spojovo-orientované (TCP, SCTP), ktoré vytvárajú a udržujú spojenie a zabezpečujú spoľahlivosť príjmu dát a bezspojové (UDP), ktoré spojenie nenadväzujú. Dátové entity transportnej vrstvy sú segmenty (anglicky segment).

4. Aplikačná vrstva TCP/IP zodpovedá 3 najvyšším vrstvám OSI modelu (aplikačná, prezenčná, relačná). Protokoly tejto vrstvy často využívajú protokoly nižších vrstiev ako čierne skrinky. Často sú spájané s aplikáciami typu klient-server, pričom pre určité aplikácie na strane servera sú vyhradené príslušné čísla portov (napríklad HTTP má port 80, Telnet má port 23). Porty slúžia na oddelenie jednotlivých komunikácií a typov týchto komunikácií prebiehajúcich na jednom zariadení. Procesy tejto vrstvy sú zavislé na konkrétnej aplikácii.

2.3 Vybrané sieťové protokoly

Z pohľadu tejto práce sú najzaujímavejšie hlavne protokoly IP, TCP a UDP, ktoré patria v dnešnej dobe medzi najrozšírenejšie. Navyše samotná klasifikácia paketov využíva v prevažnej väčšine polia práve týchto protokolov.

2.3.1 Protokol IP

Protokol IP je definovaný v [17]. Ide o protokol sieťovej vrstvy, ktorý neposkytuje žiadne mechanizmy pre riadenie toku či zaručenie spoľahlivého prenosu. Momentálne je to najrozšírenejší protokol na výmenu dát v sieťach založených na prepínaní paketov. Protokol nenadväzuje spojenia a príjemca, ktorý ani nemusí existovať, sa dozvie o komunikácii až v momente príchodu paketu. Protokol sa vyznačuje nespoľahlivým doručovaním dát (anglicky best-effort delivery), kedy sa príslušné smerovače na ceste od zdroja snažia nájsť najlepšiu cestu k príjemcovi (táto cesta je daná smerovacími protokolmi a protokol IP tomuto procesu podlieha, o kvalite tejto cesty však nie je nič garantované). Základný blok dát, s ktorým sa na úrovni IP pracuje, sa nazýva paket. Paket v sebe zahŕňa hlavičku a samotné dáta (anglicky payload).

Verzia protokolu IPv4 definuje adresu ako 32-bitové číslo, ktoré sa z dôvodu čitateľnosti rozdelí na 4 8-bitové čísla, ktoré sa následne zapíšu v desiatkovej sústave oddelené bodkami (napríklad 192.168.1.1). Adresový priestor IPv4 však bol vyčerpaný, preto sa postupne prechádza na novú verziu protokolu IPv6. IPv6 adresa je 128-bitové číslo rozdelené na 8 16-bitových čísel, ktoré sa zapisujú v šestnástkovej sústave oddelené dvojbodkami. Väčšia šírka adresy poskytuje väčší adresový priestor, ktorý by mal byť aj v budúcnosti dostačujúci.

2.3.2 Protokol UDP

Protokol UDP je jednoduchým protokolom transportnej vrstvy sieťového modelu ISO/OSI. Popis tohto protokolu je obsiahnutý v dokumente [16]. Protokol umožňuje doručovanie dát s nízkou réžiou, avšak bez garancie doručenia. Využíva nespoľahlivé doručovanie poskytované protokolom sieťovej vrstvy. Funguje v bezstavovom režime, bez toho aby vytváral

a udržoval spojenie. Základný blok dát, s ktorým sa na úrovni UDP pracuje, sa nazýva segment.

Protokol UDP teda neposkytuje žiadne mechanizmy pre spoľahlivé doručenie dát, riadenie toku ani prevenciu zahltenia. Preto je vhodný na použitie hlavne v aplikáciách, ktoré netolerujú veľké oneskorenie dát a vedia si poradiť s chýbajúcimi dátami. Ide najmä o IP telefóniu a streaming.

2.3.3 Protokol TCP

Protokol TCP je protokolom transportnej vrstvy sieťového modelu ISO/OSI. Poskytuje spoľahlivý spojovo-orientovaný prenos medzi dvoma koncovými bodmi v sieti (vytvára logické spojenie týchto dvoch bodov). Nutné požiadavky, ktoré musí implementácia daného sieťového protokolu obsahovať, je možné nájsť v dokumente [2]. Základný blok dát, s ktorým sa na úrovni TCP pracuje, sa nazýva segment. Segment zahŕňa hlavičku a samotné dáta (anglicky payload).

Keďže ide o protokol transportnej vrstvy, tvorí prechodnú úroveň medzi aplikáciou a protokolom sieťovej vrstvy (napríklad IP). Protokol IP neposkytuje žiadne prostriedky na detekciu ani opravu viacnásobného prijatia paketu, straty paketu alebo prijatia paketov v nesprávnom poradí. Protokol TCP preto definuje mechanizmy, ktoré dokážu tieto problémy detekovať a prípadne dáta znovu poslať alebo usporiadať, čím zabezpečuje spoľahlivý prenos dát medzi koncovými zariadeniami. Zároveň dokáže riadiť tok dát a tým niektorým týmto problémom predchádzať.

TCP sa v dnešnej dobe vďaka svojej spoľahlivosti využíva hlavne v aplikáciách, pri ktorých vyžadujeme správnosť prijatia dát. Cenou za spoľahlivé doručenie dát však môže byť zvýšenie oneskorenia prijatých dát, aplikácie preto musia byť schopné tolerovať takéto zvýšené oneskorenie.

Kapitola 3

Softvérovo definované sieťovanie

Informácie tejto kapitoly boli čerpané z [11]. Softvérovo definované sieťovanie (anglicky software-defined networking, SDN) je sieťová architektúra, v ktorej je kontrolná úroveň oddelená od dátovej, inteligencia siete a jej stav sú centralizované a infraštruktúra je pre aplikácie abstrahovaná. Vďaka týmto vlastnostiam je možné dosiahnuť lepšiu programovateľnosť, automatizáciu, sieťovú kontrolu a následne budovať škálovateľné a flexibilné siete.

Za rozvoj a štandardizáciu tejto architektúry momentálne zodpovedá nezisková organizácia ONF (Open Networking Foundation).

Motiváciou pre vznik SDN je nutnosť prispôsobiť sieťovú architektúru spôsobu, ktorým sa v dnešnej dobe siete využívajú. Štandardné siete sú hierarchické, budované z vrstiev prepínačov usporiadaných v stromovej štruktúre. Takáto architektúra je vhodná v prípade, že je dominantný spôsob komunikácie typu klient-server, nie je však vhodná pre dynamické potreby dnešných dátových centier a distrubuované výpočty. Trendy, ktoré viedli k vytvoreniu tejto architektúry sú:

- **Meniace sa vzory použitia sietí**. Na rozdiel od aplikácií typu klient-server, kde sa väčšina komunikácie odohráva medzi jedným klientom a jedným serverom, dnešné aplikácie (hlavne v dátových centrách) pristupujú k viacerým rôznym databázam a serverom, čím vytvárajú množstvo klasických bodových spojení, ktoré následne musia v koncovom zariadení spojiť.

- **Rozšírenie mobilných zariadení**. Mobilné zariadenia, ako notebooky, tablety a rôzne mobilné telefóny sú uživateľmi na prístup do sietí používáne stále viac.

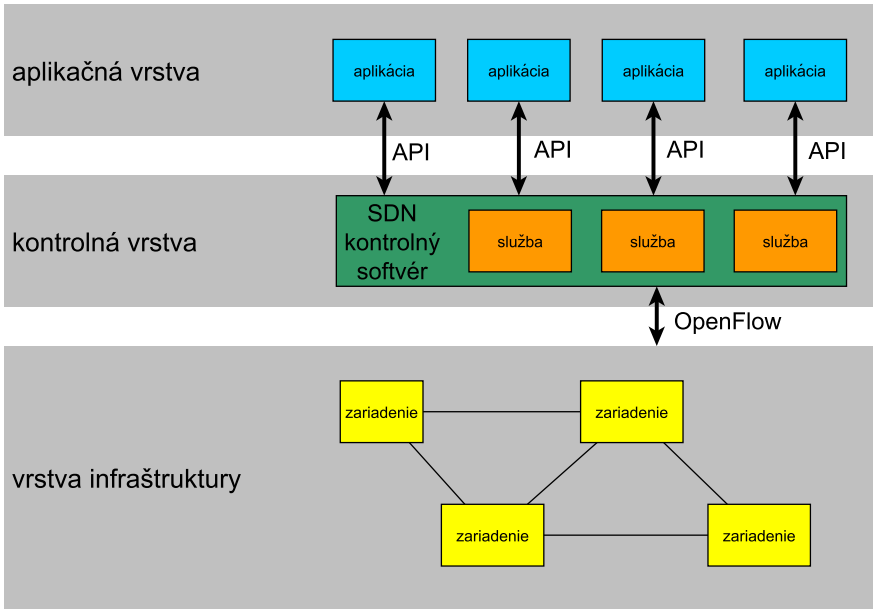
- **Veľké objemy dát**. Na spracovanie veľkých objemov dát je nutné využívať masívny paralelizmus niekedy na tisíckách serverov, čo vyžaduje veľké množstvo priamych spojení. Dátové centrá preto potrebujú škálovať siete do stále väčších veľkostí, pričom musia zachovávať možnosť spojenia každého zariadenia s každým.

Tieto trendy odhaľujú limity štandardných sieťových technológií, ktoré neboli navrhnuté aby spĺňali požiadavky kladené dnešnými sieťami. Problémy, na ktoré sa naráža, sú najmä:

- **Komplexnosť vedúca k statickosti**. V prípade, že je nutné vykonať zmenu v sieti (napríklad pridať zariadenie), je potrebné vykonať zmenu na viacerých zariadeniach a pre viac rôznych protokolov. Práca potrebná na vykonanie zmeny rastie s komplexnosťou siete, čo vedie k statickosti komplexných sietí, keďže správcovia chcú minimalizovať zásah do fungujúcej siete. Táto statickosť však kontrastuje s dynamickou povahou dnešných serverov.

- **Nekonzistencia politík**. Na správu politík, ktoré sa vzťahujú na celú sieť, je nutné pri akejkoľvek zmene konfigurovať tisíce zariadení, čo je veľmi náročné. Navyše je nutné zabezpečiť konzistenciu jednotlivých politík naprieč týmito zariadeniami.
- **Neschopnosť škálovať**. S rastom sietí rastie aj ich komplexnosť a zložitosť ich správy. Škálovateľnosť založená na očakávaných vzoroch komunikácie nie je možná, keďže tieto vzory sú stále dynamickejšie a teda nepredvídateľné.
- **Závislosť na výrobcoch**. Nové služby a schopnosti je nutné vytvárať a nasadzovať čo najrýchlejšie. Je však potrebné čakať, kým výrobca zariadenia zavedenie podporu týchto nových schopností do zariadenia, čo môže trvať aj niekoľko rokov.

Hlavná myšlienka architektúry SDN spočíva v oddelení kontroly a riadenia sieťovej infraštruktúry od funkčných prvkov infraštruktúry starajúcich sa o samotné smerovania dát. Architektúra navyše poskytuje priame programovanie riadenia siete. Oddelenie riadenia od individuálnych sieťových zariadení umožňuje zaviesť abstrakciu nad funkčnou infraštruktúrou. Aplikácie a sieťové služby sa potom môžu pozerať na sieť ako na jednotnú logickú jednotku poskytujúcu prepojenie koncových zariadení. Spomenutá abstrakcia prináša automatizáciu fungovania sieťovej infraštruktúry a zjednodušuje tvorbu škálovateľných a flexibilných sietí. Architektúru SDN ilustruje obrázok 3.1. Z logického pohľadu je architektúru možné rozdeliť na niekoľko vrstiev. Nad fyzickými zariadeniami tvoriacimi infraštruktúru siete je vytvorená abstrakná kontrolná vrstva. V tejto vrstve je inteligencia siete logicky centralizovaná v softvérových SDN kontroléroch, ktoré udržujú globálny pohľad na sieť. Pre samotné aplikácie sa teda sieť javí ako jeden logický prepínač. Navyše je možné riadiť celú sieť z jedného logického miesta a to nezávisle na špecifických vlastnostiach jednotlivých zariadení infraštruktúry. Návrh a správa siete je potom výrazne jednoduchšia. Jednoduchšie sú aj samotné zariadenia sieťovej infraštruktúry, ktoré musia rozumieť len inštrukciám od SDN kontroléru.



Obr. 3.1: Architektúra softvérovo definovaného sieťovania.

Medzi hlavné výhody, ktoré architektúra prináša, patria napríklad:

- **Centralizovaný manažment a kontrola sieťových zariadení**. Inteligencia siete je logicky centralizovaná v softvérových kontroléroch, ktoré uchovávajú globálny pohľad na sieť. Sieť sa potom aplikáciám a správcom politík javí ako jeden logický celok.
- **Zjednodušená automatizácia a využívanie spoločných aplikačných rozhraní**. Spoločné aplikačné rozhrania poskytujú vyššiu úroveň dosiahnutej abstrakcie. Sieťové zariadenia môžu byť jednoduché, keďže nemusia priamo rozumieť všetkým protokolom, ale stačí aby rozumeli inštrukciám od kontrolérov. Správcovia sietí nemusia ručne konfigurovať všetky zariadenia, ale stačí nakonfigurovať zjednodušené sieťové abstrakcie.
- **Rýchle inovovanie**. Pridávania nových schopností siete je možné bez potreby konfigurovať všetky zariadenia. SDN podporuje množinu aplikačných rozhraní, ktoré umožňujú implementovať bežné sieťové služby, ako je smerovanie, bezpečnosť, kontrola prístupu, kvalita služieb, optimalizácie, manažment politík a iné, prispôsobené kontrétnym potrebám danej siete.
- **Programovateľnosť zariadení väčšou skupinou ľudí**. Zariadenia je možné jednoduchšie programovať s využitím bežných programovacých jazykov a prostredí.
- **Zvýšená bezpečnosť a spoľahlivosť sietí**. Ide o dôsledok centralizovaného a automatizovaného manažmentu, jednotných politík a menšieho množstva chýb vznikajúcich pri konfigurácii.
- **Lepšia granularita sietí**.

Vývoj SDN priniesol aj vnik protokolu OpenFlow. OpenFlow je prvé štandardné komunikačné rozhranie medzi kontrolnou vrstvou a vrstvou infraštruktúry SDN architektúry. OpenFlow poskytuje priamy a jednotný prístup k sieťovým zariadeniam vrstvy infraštruktúry (prepínače, smerovače). Takýto protokol je potrebný, aby bolo možné oddeliť sieťovú kontrolu z prepínačov do logicky centralizovaného softvéru.

OpenFlow špecifikuje základné primitíva, používané externými softvérovými aplikáciami, na konfiguráciu smerovania na jednotlivých sieťových zariadeniach. Tento protokol je implementovaný v sieťových zariadeniach, ale aj v rámci kontrolného softvéru. OpenFlow využíva koncept tokov (pakety patriace do rovnakého spojenia, majú rovnakú päticu zdrojová IP adresa, cieľová IP adresa, zdrojový port, cieľový port a protokol) na identifikáciu sieťovej premávky na základe vopred definovaných statických a dynamických pravidiel nad tokmi alebo ich skupinami. Možnosť definovať správanie pre jednotlivé toky umožňuje dosiahnuť jemnú granularitu siete.

SDN využívajúce OpenFlow je využívané na reálnych sieťach (napríklad firmou Google). Túto technológiu je možné použiť aj paralelne s tradičným smerovaním, čo zjednodušuje prechod na SDN a zároveň umožňuje využívať SDN len nad časťou siete, prípadne v sieťach, v ktorých niektoré koncové zariadenia SDN nepodporujú.

Kapitola 4

Jazyk P4

Informácie pre túto kapitolu boli čerpané z článku [1] a zo špecifikácie [12]. Kapitola začína základnou motiváciou a cieľmi jazyka P4, pokračuje popisom abstraktného modelu, na ktorom je jazyk P4 založený a nakoniec popisuje samotný jazyk a jeho konštrukcie.

Motiváciou pre vznik jazyka P4 je vznik rôznych nových protokolov, ktoré majú za následok nutnosť neustále rozširovať špecifikáciu architektúr ako SDN s využitím OpenFlow. OpenFlow bolo v pôvodnej verzii abstrakciou jednej tabuľky pravidiel pre 12 rôznych polí hlavičiek. Postupne je však túto abstrakciu nutné rozširovať o ďalšie polia a ďalšie tabuľky. Vo verzii 1.4 už OpenFlow podporuje 41 polí. Tento trend pridávania polí a tabuliek by mal v budúcnosti pokračovať. Namiesto neustáleho rozširovania špecifikácie OpenFlow je preto vhodnejšie podporovať flexibilnejšie mechanizmy na analýzu paketov a porovnávanie polí hlavičiek. Práve o to sa jazyk P4 snaží.

Jazyk P4 má slúžiť na konfiguráciu prepínačov, konkrétne prepínaču hovorí akým spôsobom má spracovávať pakety. Tým zvyšuje úroveň abstrakcie, na ktorej je možné siete programovať. Je však nutné nájsť správny pomer medzi vyjadrovacou schopnosťou jazyka a zložitosťou implementácie nad celou škálou rôznych softvérových a hardvérových zariadení.

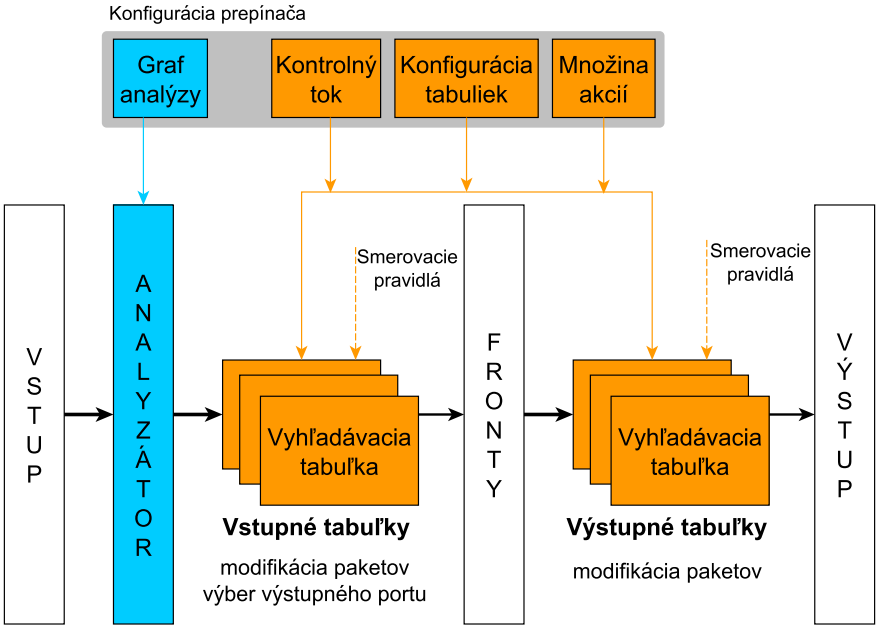
Jazyk P4 má stanovené 3 hlavné ciele:

- **Schopnosť rekonfigurácie**. Kontrolér by mal byť schopný meniť spôsob spracovania paketov počas behu.
- **Nezávislosť na protokole**. Prepínač by nemal byť závislý na konkrétnom formáte paketov. Kontrolér má byť schopný špecifikovať analyzátor hlavičiek paketov a množinu vyhľadávacích tabuliek, ktoré tieto hlavičky spravujú.
- **Nezávislosť na cieľovej architektúre**. Programátor kontroléra nemusí poznať cieľovú architektúru prepínača. Prekladač jazyka by mal byť schopný preložiť cieľovo nezávislý popis (popis v jazyku P4) na cieľovo závislý program (konfigurácia prepínača).

4.1 Abstraktný smerovací model

Abstraktný model jazyka P4 je na obrázku 4.1. Prepínač najprv analyzuje hlavičky paketov a následne na základe získaných hodnôt polí hlavičiek vykoná spracovanie paketu v podobe aplikácie niekoľkých vyhľadávacích tabuliek. Oproti OpenFlow obsahuje model analyzátor hlavičiek, ktorý je programovateľný, vyhľadávacie tabuľky pravidiel môžu byť

navzájom paralelné a akcie, ktoré sa nad paketom vykonajú, sú tvorené z primitív podporovaných prepínačom. Model generalizuje ako sú pakety spracované na rôznych zariadeniach a technológiach.



Obr. 4.1: Abstraktný smerovací model jazyka P4.

Model je riadený dvoma druhmi operácií. Konfiguračné operácie programujú analyzátor hlavičiek, poradie aplikovaných vyhľadávacích tabuliek a špecifikujú formát hlavičiek protokolov. Druhý typ operácií napĺňa vyhľadávacie tabuľky.

V abstraktnom modele sú prichádzajúce pakety najprv spracované analyzátorom hlavičiek, ktorý z nich extrahuje jednotlivé potrebné polia. Význam jednotlivých polí je ignorovaný, analyzátor zaujíma len hodnota poľa.

Vyextrahované polia sú následne odovzdané vyhľadávacím tabuľkám. Tieto tabuľky reprezentujú množinu pravidiel, na základe ktorých sa pakety spracúvajú. Tabuľky sú rozdelené do dvoch skupín, na vstupné a výstupné tabuľky. Obe skupiny tabuliek môžu paket modifikovať zmenou hlavičky, vstupné tabuľky navyše určujú výstupný port paketu a front, do ktorého sa daný paket zaradí pred ďalším spracovaním. Na základe vyhľadania vyextrahovaných hodnôt polí hlavičiek paketu v množine pravidiel, ktoré tabuľka reprezentuje, je vybratá akcia, ktorá sa na paket aplikuje. Výsledkom tejto akcie môže byť smerovanie, duplikovanie alebo zahodenie paketu.

Na prenos ďalších potrebných informácií medzi jednotlivými krokmi spracovania paketu slúžia metadáta, ktoré sú považované za ekvivalent dát hlavičiek. Príkladom takýchto metadát je napríklad vstupný port, z ktorého bol paket prijatý, časová značka a iné.

4.2 Programovací jazyk

Na základe abstraktného modelu je definovaný samotný jazyk P4. Jazyk obsahuje základné koncepty na definíciu formátu hlavičiek a metadát, popis analyzátora hlavičiek, definíciu

vyhľadávacích tabuliek, definíciu akcií na základe primitív a na definíciu kontrolného toku, ktorý udáva rozloženie vyhľadávacích tabuliek.

Na ukladanie stavu na dobu dlhšiu ako spracovanie jedného paketu navyše jazyk podporuje definície rôznych počítadiel a meradiel. Prístup k týmto počítadlám môžu mať povelený len niektoré vyhľadávacie tabuľky.

Protokoly, s ktorými sa pracuje, sú plne v kompetencii programátora. Jazyk P4 nepozná žiadne štandardné protokoly, preto ako prvý krok pri vytvorení P4 programu je nutné štruktúru hlavičiek použitých protokolov definovať. Hlavička je definovaná ako postupnosť jednotlivých polí, ktoré túto hlavičku tvoria. Polia majú definovanú šírku v bitoch (jedno pole v rámci jednej hlavičky môže mať premenlivú šírku), saturáciu (implicitne je vypnutá) a znamienkovosť hodnoty.

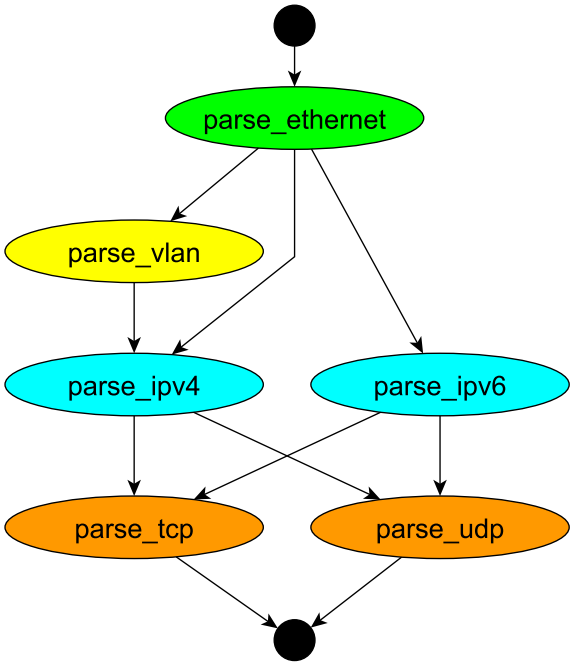
Analyzátor paketov z abstraktného modelu z predchádzajúcej sekcie zabezpečuje prvý krok spracovania každého paketu a je založený na stavovom automate. Tento stavový automat je reprezentovaný grafom prechodov, tiež nazývaným graf analýzy (anglicky parse graph), ktorý vyjadruje poradie hlavičiek v pakete. Prepínač podporujúci jazyk P4 teda musí byť schopný implementovať stavový automat a musí dokázať prechádzať obsah hlavičiek paketu po bitoch.

```
1  parser parse_ethernet {
2    extract(ethernet);
3    return select(latest.ethertype) {
4      0x8100 : parse_vlan;
5      0x800 : parse_ipv4;
6      ...
7      default : parse_def;
8    }
9  }
10 parser parse_ipv4 {
11   extract(ipv4);
12   return select(latest.protocol) {
13     PROTO_TCP : parse_tcp;
14     PROTO_UDP : parse_udp;
15     ...
16   }
17 }
18 parser parse_udp {
19   extract(udp);
20   return ingress;
21 }
```

Kód 4.1: Definícia analyzátora paketov ako grafu prechodov v jazyku P4.

V jazyku P4 môže programátor definovať štruktúru analyzátora priamo popisom prechodov grafu analýzy (tieto prechody vyjadrujú následnosti, v ktorých sa jednotlivé hlavičky spracovávajú). Každý prechod môže byť navyše podmienený napríklad hodnotou vybraného poľa momentálne spracovanej hlavičky. Vyextrahované hodnoty jednotlivých polí hlavičiek sa ukladajú do špeciálnej vnútornej reprezentácie (anglicky parsed representation), ktorá sa následne predáva tabuľkám.

Príklad definície jednoduchého analyzátora je v kóde 4.1, zodpovedajúca abstraktná reprezentácia v podobe grafu prechodov je na obrázku 4.2. Ilustrovaný graf prechodov



Obr. 4.2: Príklad grafu prechodov popisujúceho analyzátor paketov.

popisuje funkciu analyzátora a funguje jednoducho. Ako prvá je spracovaná ethernetová hlavička. Na základe hodnoty poľa ethernetovej hlavičky obsahujúceho označenie protokolu nasledujúcej hlavičky sa prejde na spracovanie tohto protokolu sieťovej vrstvy (IPv4, IPv6). Následne sa opäť na základe hodnoty poľa obsahujúceho kód nasledujúceho protokolu prejde na spracovanie daného protokolu transportnej vrstvy (UDP, TCP).

Po tom, čo sú spracované hlavičky prijatého paketu, je nutné paket ďalej spracovať. Na samotné spracovanie paketov musí programátor definovať vyhľadávacie tabuľky. Vyhľadávacie tabuľky predstavujú reprezentáciu klasifikátora, respektíve množiny klasifikačných pravidiel. Definícia vyhľadávacej tabuľky pravidiel v jazyku P4 obsahuje hlavne zoznam polí, na základe ktorých sa má vyhľadanie pravidla vykonať (teda dimenzie klasifikácie). Pre každé pole je definovaný typ zhody, ktorá sa kontroluje. Definícia vyhľadávacej tabuľky navyše obsahuje množinu akcií, ktoré môžu byť na paket aplikované a prípadne maximálnu veľkosť tabuľky. Jazyk P4 zatiaľ špecifikuje 4 typy zhody:

- **Presná zhoda (anglicky exact)**. Porovnávané hodnoty poľa paketu a referenčnej hodnoty obsiahnutej v pravidle musia byť rovnaké.
- **Hodnota je v rozsahu (anglicky range)**. Pravidlá definujú rozsah, kontroluje sa, či je hodnota daného poľa z hlavičky paketu v danom rozsahu.
- **Ternárna (anglicky ternary)**. Je použitá maska, ktorou sa niektoré bity porovnávaných hodnôt vymaskujú (pomocou bitového súčinu hodntoy a masky) a až následne sa kontroluje či sú hodnoty rovnaké.
- **Najdlhší zhodný prefix (anglicky longest prefix match, LPM)**. Hľadá sa pravidlo, ktoré obsahuje referenčnú hodnotu, ktorá má s hodnotou získanou z hlavičky paketu čo najdlhší zhodný bitový prefix.

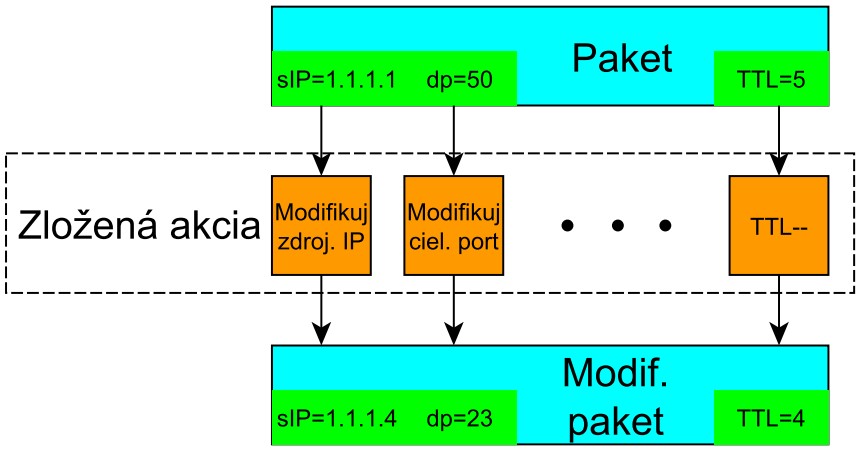
Každý riadok tabuľky, ktorý reprezentuje jedno pravidlo, by potom mal obsahovať aspoň hodnoty polí, s ktorými sa zhoda porovnáva, odkaz na akciu, ktorá sa v prípade výberu daného riadku (pravidla) vykoná a parametre, s ktorými je akcia vykonaná.

Ako je tabuľka implementovaná nie je definované, výsledná implementácia je kompletne v kompetencii prekladača.

```
1  table filter_flow {
2    reads {
3      network_header.src_addr : exact;
4      network_header.dst_addr : exact;
5      transport_header.src_port : exact;
6      transport_header.dst_port : exact;
7      transport_header.protocol : exact;
8    }
9    actions {
10     drop;
11   }
12   size 16384;
13 }
```

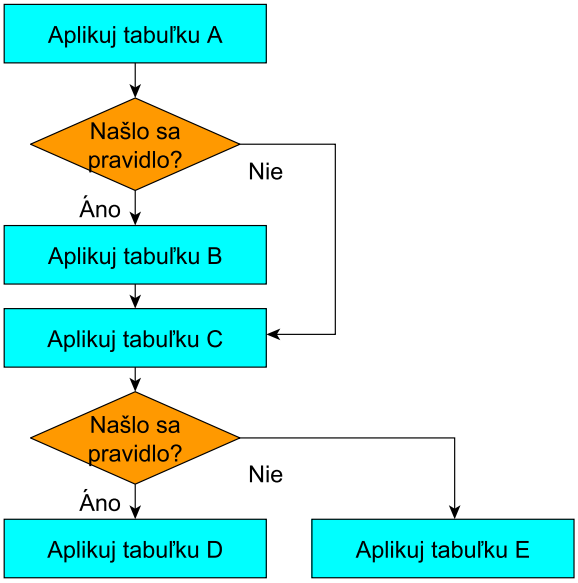
Kód 4.2: Príklad zápisu vyhľadávacej tabuľky v jazyku P4.

Kód 4.2 ilustruje zápis jednoduchej tabuľky slúžiacej na filtráciu paketov. Tabuľka filtruje na základe pätice identifikujúcej tok. Jednotlivé pravidlá typu blokuj tento tok sú pridávané až počas behu. Pri nájdení pravidla sa môže vykonať len jedna akcia a to je akcia *drop*, ktorá zahodí paket.



Obr. 4.3: Ilustrácia fungovania zložených akcií jazyka P4.

Ďalším krokom programátora je definovanie zložitejších akcií, ktoré jednotlivé tabuľky môžu využívať. Princíp fungovania zložitejších akcií ilustruje obrázok 4.3 Jazyk P4 obsahuje množinu primitívnych akcií, z ktorých je možné skladať zložitejšie akcie. Jazyk špecifikuje primitíva na modifikáciu polí, pridávanie a odstraňovanie hlavičiek, operácie nad stavovou pamäťou, klonovanie paketov a ďalšie ako napríklad počítanie kontrolných súčtov. P4 predpokladá paralelné vykonávanie jednotlivých primitív v rámci jednej zloženej akcie. Prepínače neschopné paralelného spracovania môžu túto sémantiku vhodne emulovať.



Obr. 4.4: Jednoduchý príklad popisu kontrolného toku pomocou grafu.

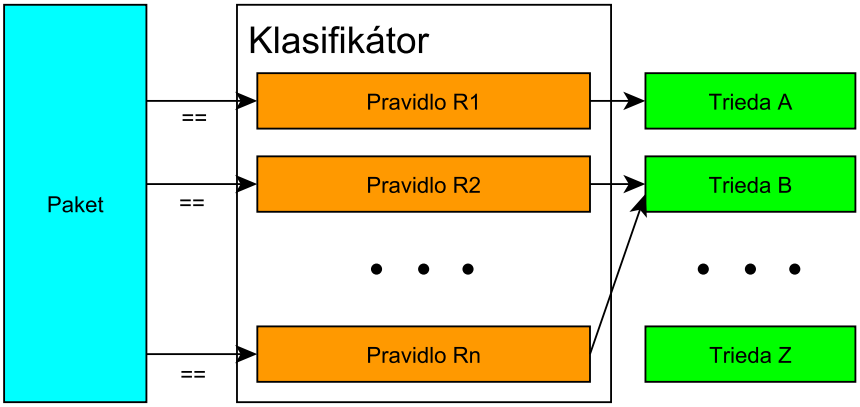
Po tom, čo sú definované tabuľky a akcie, musí programátor ešte definovať akým spôsobom sa bude odovzdávať kontrola jednotlivým tabuľkám. V podstate teda definuje ako sú tabuľky usporiadané. Situáciu demonštruje obrázok 4.4. Na tento účel slúži popis kontrolného toku. Ide o imperatívny popis obsahujúci množinu funkcií, podmienok a odkazov na tabuľky. Ide hlavne o sekvenciu aplikovaných tabuliek, pričom aplikácie niektorých tabuliek môžu byť podmienené úspešným vyhľadaním v predchádzajúcej tabuľke. Aplikácia tabuľky znamená vyhľadanie pravidla a vykonanie akcie súvisiacej s nájdeným pravidlom.

Jazyk P4 je v čase písania tejto práce ešte stále vo vývoji. Existuje prvá verzia špecifikácie. Časti jazyka sa preto ešte stále môžu meniť. Jazyk však stále naberá na popularite a postupne sa do jeho vývoja zapájajú veľké firmy (napríklad Google, Intel a ďalší). Preto je vysoko pravdepodobné, že sa v budúcnosti bude tento jazyk využívať.

Kapitola 5

Klasifikácia paketov

Informácie tejto kapitoly boli čerpané najmä z článkov [6], [27], [7]. Jazyk P4 sa snaží o obecné programovanie OpenFlow prepínačov, ktoré sú v dnešnej dobe stále rozšírenejšie. Jednou z najdôležitejších častí jazyka P4 sú vyhľadávacie tabuľky. Funkcionalita, ktorú tieto tabuľky v rámci jazyka P4 zabezpečujú, predstavuje priamo klasifikáciu paketov. Jedná sa o proces, ktorý je výpočetne relatívne náročný, a preto je na vyriešenie rýchleho spracovania vyhľadávacích tabuliek, potrebné zaoberať sa klasifikáciov paketov a algoritmami, ktoré sa pri klasifikácii využívajú.



Obr. 5.1: Princíp procesu klasifikácie.

Proces klasifikácie paketov bližšie ilustruje obrázok 5.1. Cieľom klasifikácie paketov je roztriediť pakety do určitých tried na základe aplikácie množiny pravidiel. Reálne to znamená nájsť pravidlo z množiny pravidiel, s ktorým sa vybrané informácie z hlavičky paketu zhodujú. Pokiaľ je takýchto pravidiel viac, je vhodné vybrať pravidlo s najväčšou prioritou.

Množinu pravidiel R tiež označujeme ako klasifikátor. Každé pravidlo R_i z množiny pravidiel R pozostáva z hodnoty, nad ktorou sa vykonáva zhoda s hodnotou získanou z hlavičky paketu. Pravidlo navyše odkazuje na niektorú z výsledných tried, prípadne akcií, ktoré sa majú vykonať. Najčastejšie ide o jednoduchú akciu ako napríklad zahodenie paketu.

Hodnota, s ktorou sa pole hlavičky paketu porovnáva, môže mať všeobecne tvar regulárneho výrazu. V praxi je často implementácia porovnávania s regulárnym výrazom zbytočne náročná, preto býva regulárny výraz obmedzený na jednoduchú hodnotu, rozsah hodnôt, masku alebo bitový prefix.

Princíp klasifikácie je možné rozšíriť tak, že sa pre každý paket berie do úvahy viac rôznych polí hlavičky. V takomto prípade hovoríme o viacdimenzionálnej klasifikácii. Jednotlivé polia predstavujú dimenzie klasifikácie a ich počet je teda počet dimenzií klasifikácie. Pravidlá sú potom reprezentované ako n-tice hodnôt, s ktorými sa porovnáva.

P	cieľová IP	Zdrojová IP	cieľový port	Prot.	Akcia
1	121.155.142.10	121.155.80.5	*	*	blokuj
2	121.150.*.*	*	*	TCP	blokuj
3	121.150.*.*	*	53	UDP	blokuj
4	*	117.18.8.*	*	IP	blokuj
5	*	*	*	*	povoľ

Tabuľka 5.1: Príklad pravidiel klasifikátoru.

Paket	cieľová IP	Zdrojová IP	cieľový port	Prot.	Pravidlo	Akcia
1	121.155.142.10	121.155.80.5	1028	UDP	1	blokuj
2	121.151.182.78	117.18.8.17	2053	IP	4	blokuj
3	77.45.124.5	201.25.48.15	1353	TCP	5	povoľ

Tabuľka 5.2: Príklad vybraných pravidiel pre pakety.

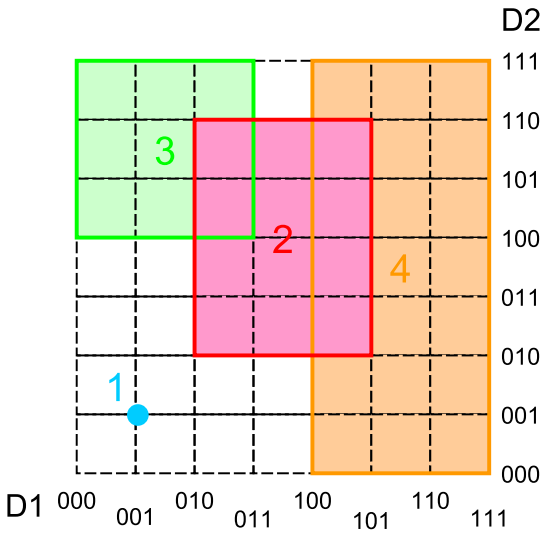
Pravidlo	D1 (3b)	D2 (3b)
1	001	001
2	010-101	010-110
3	000-011	100-111
4	100-111	000-111

Tabuľka 5.3: Jednoduchý klasifikátor pre ilustráciu geometrickej reprezentácie.

Tabuľka 5.1 ilustruje jednoduchý klasifikátor slúžiaci na blokovanie niektorých vybraných paketov. Tabuľka 5.2 potom ukazuje akciu a pravidlo, ktoré je aplikované na konkrétne prichádzajúce pakety. Klasifikátor je možné reprezentovať aj geometricky. Pri geometrickej reprezentácii predstavujú dimenzie jednotlivé osi priestoru a pravidlá sú viac rozmernými telesami v tomto priestore, prípadne bodmi v prípade, že pravidlo reprezentuje zhodu s konkrétnou hodnotou nie s rozsahom. Jednoduchý dvojdimenzionálny klasifikátor z tabuľky 5.3 je geometricky reprezentovaný obrázkom 5.2.

Na posudzovanie jednotlivých algoritmov klasifikácie je dobré zamerať sa na niektoré významné vlastnosti. Medzi tieto vlastnosti patrí:

- **Rýchlosť vyhľadania**. Čím ďalej dôležitejšia vlastnosť, keďže rýchlosti na ktorých siete operujú neustále rastú. Pre priepustnosť 10Gbps môže byť v hraničnom prípade, kedy budú TCP pakety dlhé 40 bajtov, nutné spracovať až vyše 31 miliónov paketov za sekundu.
- **Nízka pamäťová náročnosť**. Nízka pamäťová náročnosť umožňuje využitie menších a rýchlejších pamätí, môže teda ovplyvniť rýchlosť vyhľadania.
- **Schopnosť pracovať s veľkými klasifikátormi**. Komplexnejšie a väčšie siete môžu vyžadovať veľké množstvo pravidiel.



Obr. 5.2: Geometrická reprezentácia klasifikátora.

- **Rýchle úpravy množiny pravidiel**. V reálnom nasadení sa množina pravidiel počas behu mení, je preto potrebné byť schopný čo najrýchlešie pravidlá pridávať a odoberať. Pri niektorých algoritmoch to znamená budovanie celej dátovej štruktúry od začiatku.
- **Škálovatelnosť na veľký počet dimenzií**. Niektoré algoritmy narastajú v zložitosti neúmerne počtu dimenzií.
- **Flexibilita**. Algoritmy by mali podporovať rôzne typy zhody v jednotlivých dimenziách, vrátane menej bežných ako sú nespojité masky, rôzne operátory typu väčší, menší a podobne.

Algoritmy je možné rozdeliť podľa počtu dimenzií na jedno dimenzionálne a všeobecne n-dimenzionálne. Algoritmy pre vyhľadanie pravidiel sú založené na rôznych technikách, napríklad na stromových štruktúrach, dekompozícii, technike rozdeľuj a panuj, bitových vektoroch, hašovaní či karteziánskych súčinoch.

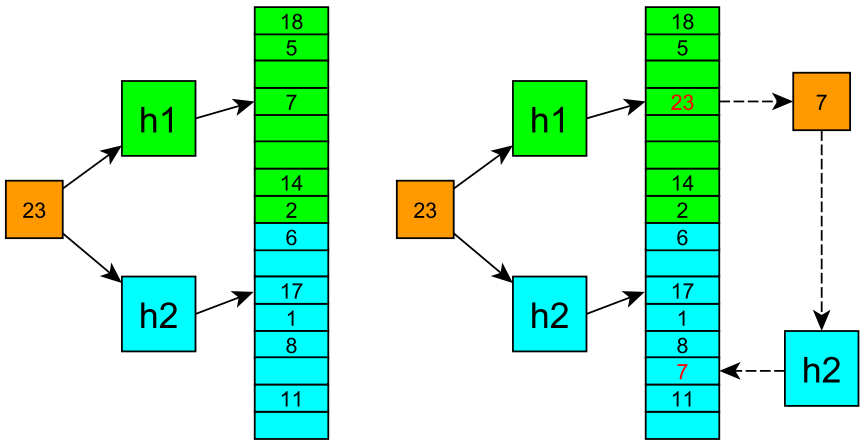
5.1 Lineárny algoritmus

Najjednoduchší algoritmus na vyhľadanie pravidla je lineárny algoritmus. Tento algoritmus jednoducho prejde postupne sekvenčne všetky pravidlá, až kým nenarazí na pravidlo, ktoré vyhovuje momentálne spracovávanému paketu.

Výhodou tohto algoritmu je jeho jednoduchosť a rýchlosť v prípade, keď je množina pravidiel extrémne malá. Pre veľké množiny pravidiel je tento spôsob riešenia veľmi neefektívny, časová zložitosť je z triedy *O(N*D)*, kde N predstavuje počet pravidiel a D predstavuje počet dimenzií.

5.2 Kukučie hašovanie

Informácie o kukučom hašovaní boli čerpané hlavne z knihy [15]. Pokiaľ pravidlá obsahujú iba presnú zhodu, je najvýhodnejšie použiť tabuľku s rozptýlenými prvkami. Ide o jednoduchú tabuľku hodnôt, ktorú ale adresujeme výsledkom špeciálnej hašovacej funkcie nad hodnotou, ktorú do tabuľky vkladáme, respektíve ktorú v tabuľke vyhľadávame.



Obr. 5.3: Princíp kukučieho hašovania a vkladania novej hodnoty do tabuľky.

Kukučie hašovanie, ilustrované obrázkom 5.3, ktorý navyše zobrazuje pridanie novej hodnoty do tabuľky, používa všeobecne viac hašovacích funkcií, štandardne dve alebo štyri, pre iné hodnoty dochádza k neefektívnemu využitiu pamäte. Tabuľka je rozdelená na viac nezávislých častí, každá časť zodpovedá jednej hašovacej funkcii. Vyhľadanie hodnoty v tabuľke je jednoduché, stačí najprv vypočítať hašovacie funkcie nad hľadanou hodnotou. Výsledky vypočítaných hašovacích funkcií udávajú index v rámci časti tabuľky, ktorá je priradená k danej funkcii. Nakoniec stačí overiť, či sa hľadaná hodnota nachádza na jednom z týchto indexov. Pridanie hodnoty do tabuľky funguje podobne. Najprv sa vykoná výpočet hašovacích funkcií, čím sa získajú indexy. Následne sa skontroluje, či je niektoré z miest daných indexami voľné. Nakoniec sa hodnota na jedno z prázdnych miest vloží. V prípade, že sú všetky takéto miesta zaplnené, je jedno z nich vybrané (náhodne alebo na základe priorít), hodnota je doň vložená nasilu a pôvodná hodnota je vytlačená (odtiaľ berie hašovanie aj svoj názov, kedže sa tento spôsob pridávania podobá kukučke, ktorá kladie svoje vajíčka do hniezd iných vtákov, pričom originálne vajíčko vytlačí z hniezda). Vytlačenú hodnotu je následne potrebné do tabuľky znovu vložiť rovnakým spôsobom (hodnota sa však nemôže vrátiť na miesto, z ktorého bola vytlačená). Kukučie hašovanie má síce priemerný čas vloženia prvku konštantný, avšak v najhoršom prípade až logaritmický. Tento jav je však možné odstrániť a to pridaním malej pamäte, ktorá slúži ako front vytlačených hodnôt. Tieto hodnoty potom nie je nutné znovu vložiť v rámci spracovania momentálne vkladanej hodnoty, ale je tak možné vykonať niekedy neskôr. Pri vyhľadaní hodnoty sa skontroluje jednak tabuľka a jednak front vytlačených hodnôt (pri vhodnej implementácii pamäte je tento čas konštantný). Využitie takejto pamäte je popísané v článku [10].

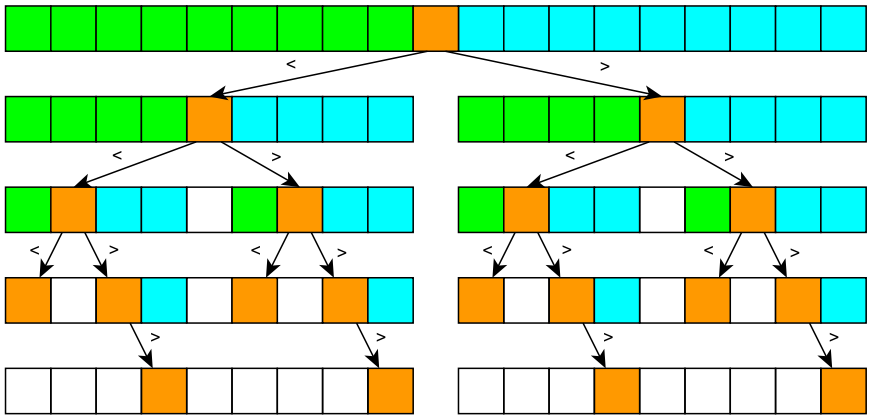
Problém nastáva, keď znovuvkladanie vytlačenej hodnoty vedie k vytlačeniu ďalšej hodnoty, ktorej znovuvloženie opäť vytláča inú hodnotu a tak ďalej, až sa dostaneme k vytlačeniu pôvodnej hodnoty. Vzniká teda akýsi nekonečný cyklus znovuvkladania. Kontolovanie

vzniku takéhoto nekonečného cyklu je možné zjednodušiť na obmedzenie maximálneho počtu znovuvložení. V prípade detekcie vzniku nekonečného cyklu je nutné tabuľku preskladať s novými hašovacími funkciami alebo je tabuľka považovaná za plnú.

Nevýhoda takéhoto riešenia je hlavne v tom, že podporuje iba presnú zhodu hodnôt. Navyše preskladanie tabuľky v prípade nekonečného cyklu znovuvkladania je výpočtovo príliš náročné. Ignorovanie vloženia hodnoty vedúcej na nekonečný cyklus znovuvkladania zasa môže viesť v extrémnych prípadoch k neefektívnemu využitiu pamäte.

5.3 Binárne vyhľadávanie

Princíp binárneho vyhľadávania zobrazuje obrázok 5.4. Cieľom je vyhľadať hodnotu v zoradenom poli hodnôt. Binárne vyhľadávanie pracuje rekurzívne. V každom kroku sa delí časť poľa, s ktorou sa pracuje, na polovice. Zároveň porovná vyhľadávanú hodnotu s mediánom momentálne delenej časti. Ak je medián väčší, pokračuje vyhľadávanie rekurzívne v prvej polovici, pokiaľ je medián menší, pokračuje sa v druhej polovici. Pravdaže v prípade rovnosti mediánu a vyhľadávanej hodnoty algoritmus úspešne končí. Algoritmus takto iteruje až kým nenašiel hľadanú hodnotu alebo kým už nemôže momentálne spracovávanú časť poľa ďalej deliť (ide teda o jednoprvkovú časť).



Obr. 5.4: Princíp binárneho vyhľadávania.

Algoritmus je schopný vyhľadať hodnotu v logaritmickom čase. Problémom však je, že pole hodnôt musí byť zoradené. Pri pridávaní a odoberaní pravidiel je preto nutné udržovať pole zoradené.

Binárne vyhľadávanie sa navyše nedá použiť na klasifikáciu priamo. Hodnoty, nad ktorými sa vyhľadáva, totiž môžu byť rôznorodé. Jednou z možností je previesť všetky možné hodnoty na rozsahy. Rozsahy vieme reprezentovať spodnou a hornou hranicou. Tieto hranice sú už presné čísla, nad ktorými vieme po zoradení vykonať binárne vyhľadávanie. Tento princíp je bližšie popísaný v článku [13].

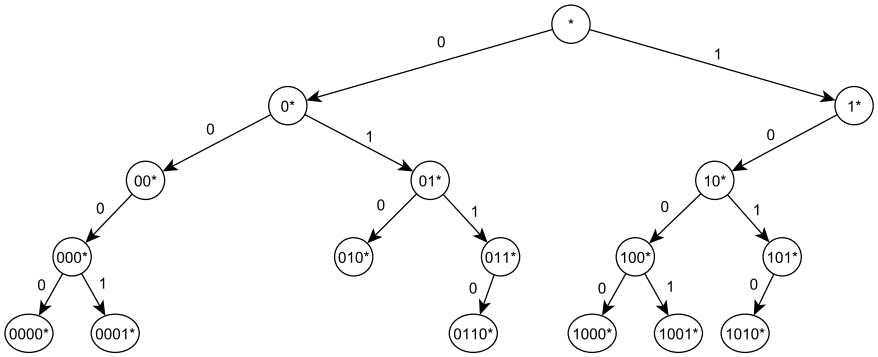
Alternatívne je možné previesť hodnoty do formy prefixov, tieto prefixy zoradiť podľa dĺžky a vykonať binárne vyhľadávanie nad rôznymi dĺžkami prefixov. Viac informácií je možné nájsť v článku [23].

Problémom binárneho vyhľadávania je však práve nevyhnutnosť zoradeného poľa. Najmä v hardvérovej implementácii môže byť radenie časovo náročné. Vhodnosť využitia binár-

neho vyhľadávania je preto obmedzená na prípady, keď dochádza k minimálnemu počtu pridaní a odobraní pravidiel.

5.4 Trie

Trie predstavuje binárnu stromovú dátovú štruktúru, ktorá je vhodná hlavne na klasifikáciu v rámci jednej dimenzie. Pravidlá sú prevedené do formy bitových prefixov a jednotlivé prefixy sú reprezentované cestami v strome. Pri vyhľadaní sa spracúva hľadaná hodnota po bitoch a v každom kroku sa na základe hodnoty bitu algoritmus posunie buď do pravého alebo ľavého potomka momentálneho uzlu. Tento postup sa opakuje, kým je to možné. Uzol, v ktorom sa skončí (respektívne celá cesta doň) predstavuje najdlhší nájdený prefix. Pridávanie a odoberanie pravidiel je taktiež jednoduché, ide v podstate o pridávanie alebo odoberanie uzlov na ceste danej pridávaným alebo odoberaným prefixom. Stromovú štruktúru bližšie ilustruje obrázok 5.5. Na obrázku sú v uzloch napísané bitové prefixy, ktoré dané uzly reprezentujú.



Obr. 5.5: Príklad stromovej štruktúry trie.

V základnej podobe algoritmus nie je veľmi efektívny, keďže pre široké dimenzie (napríklad IPv6, ktorá má až 128 bitov) je nutné urobiť v najhoršom prípade veľa krokov (pre IPv6 je to 128).

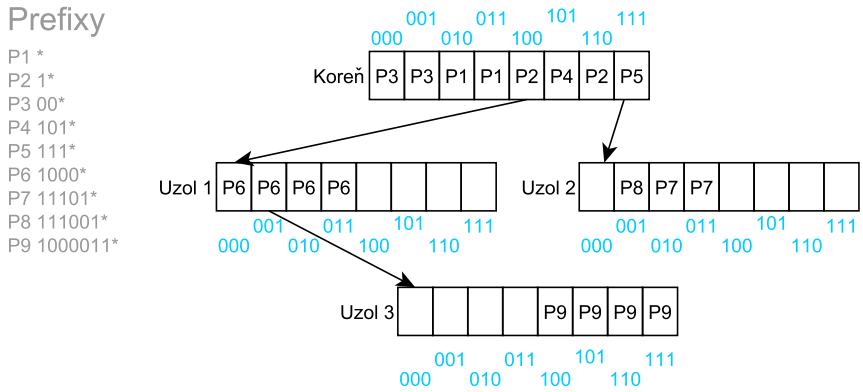
Jedno z rozšírení pre trie je viacbitové trie. Ide o rozšírenie binárneho stromu na všeobecne n-árný strom. V každom cykle algoritmu sa naraz spracúva väčšie množstvo bitov (počet spracovaných bitov sa nazýva krok). Pre správne fungovanie je nutné zarovnávať prefixy na násobky kroku. Pre krok 3 je teda napríklad potrebné upraviť prefix *0** na prefixy *000**, *001**, *010** a *011**.

Táto úprava prináša zvýšenie rýchlosti vyhľadania, ale zároveň zvyšuje pamäťové nároky.

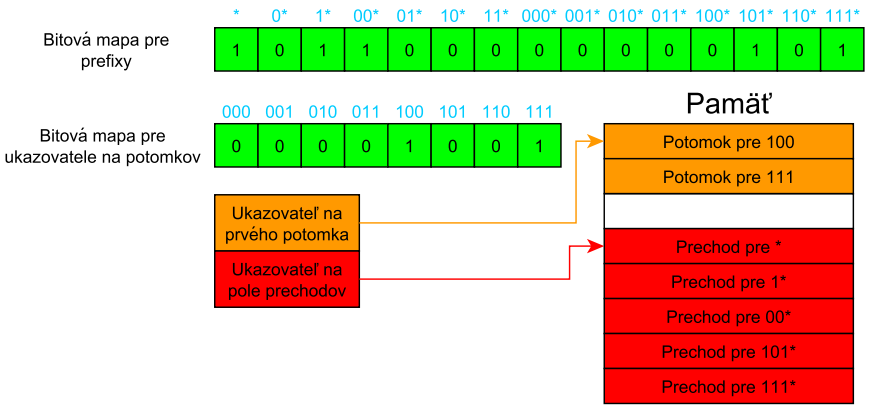
Algoritmus kontrolovanej expanzia prefixov ([20]) okrem viacbitového trie využíva aj ďalšie pamäťové optimalizácie (napríklad presúvanie prefixov medzi listami, anglicky leaf pushing). Algoritmus Lulea ([4]) ďalej vylepšuje využitie pamäte použitím kompresie pomocou bitových máp. Bitová mapa umožňuje v princípe komprimovať postupnosť $R_1R_1R_1R_2R_2R_1R_1R_1$ na postupnost $R_1R_2R_1$ a bitovú mapu *10010100* (bit 1 značí zmenu, bit 0 značí opakovanie). LC-Trie algoritmus ([14]) komprimuje viacbitové trie pomocou rekurzívneho výberu kroku, čím zabezpečí, že každý list obsahuje skutočné prefixy (nie rozšírené). Tým sa vyhýba nutnosti presúvať prefixy medzi listami.

Algoritmus Tree Bitmap ([5]) je porovnateľný s algoritmom Lulea avšak navyše poskytuje možnosť rýchleho pridávania a odoberania pravidiel.

Princíp algoritmu spočíva v tom, že uzol trie by mal mať dve funkcie, jednou je smerovanie vyhľadávania a druhou je preposlanie získanej informácie o nájdenom pravidle. Na to uzol využíva dve bitové mapy, jedna bitová mapa slúži na ukladanie vnútorných prefixov patriacich danému uzlu a druhá slúži na ukladanie ukazovateľov na svojich potomkov. Navyše algoritmus znižuje počet ukazovateľov na potomkov ukladaním potomkov rovnakého uzla spojito za sebou. Celkovo teda každý uzol uchováva len dve bitové mapy, ukazovateľ na prvého potomka a ukazovateľ na pole, ktoré obsahuje informácie o prechodoch do týchto potomkov. Algoritmus ilustrujú obrázky 5.6 a 5.7. Obrázok 5.6 ukazuje štruktúru vytvoreného viacbitového trie, zatiaľ čo obrázok 5.7 ilustruje štruktúru uložených bitových máp pre koreňový uzol. Tento algoritmus je vhodný hlavne pre hardvérové implementácie.



Obr. 5.6: Príklad vytvorenej štruktúry viacbitového trie.



Obr. 5.7: Štruktúra uložených bitových máp algoritmu Tree Bitmap pre koreňový uzol z obrázka 5.6.

O zrýchlenie algoritmu Tree Bitmap sa snaží algoritmus Hash-Tree Bitmap ([22]). Tento algoritmus využíva hašovacie funkcie ako skratky v riedkych častiach trie a klasický Tree Bitmap pre husté časti.

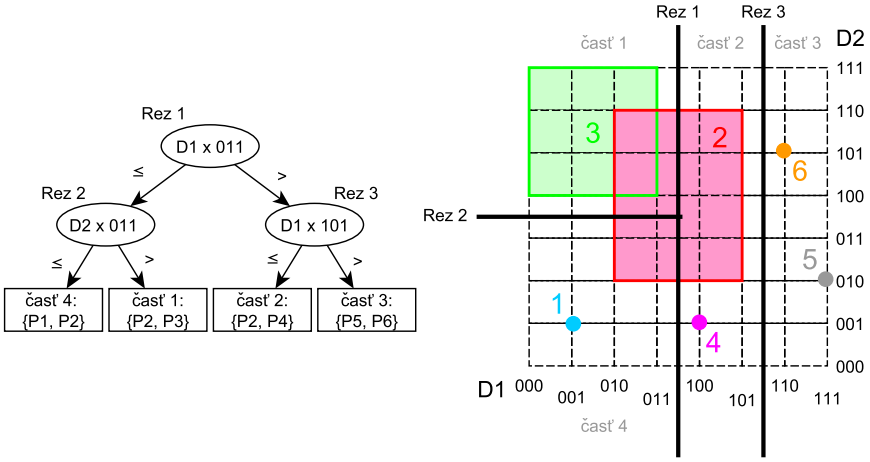
5.5 Rozšírenie algoritmov na viac dimenzií

Na rozšírenie jedno dimenzionálnych postupov na klasifikáciu viacerých dimenzií existuje škála rôznych algoritmov. Tieto algoritmy väčšinou rozširujú stromovú štruktúru trie, alebo fungujú na princípe dekompozície problému na jednotlivé dimenzie.

Algoritmus HiCuts (skratka z anglického Hierarchical Intelligent Cuttings, popísaný v článku [8]) delí prehľadávaný viacdimenzionálny priestor pomocou vhodnej heuristiky, čím delí aj množinu pravidiel na menšie podmnožiny. Na to využíva špeciálnu stromovú štruktúru. Listy tejto štruktúry obsahujú malý počet pravidiel, medzi ktorými je možné sekvenčné dohľadanie.

Každý uzol stromovej štruktúry reprezentuje časť prehľadávaného priestoru. Koreňový uzol predstavuje celý prehľadávaný priestor, ktorý je postupne delený na menšie časti reprezentované jeho potomkami. Delenie priestoru na menšie časti prebieha rezaním jednej z dimenzií na polovice (prípadne štvrtiny, osminy a tak ďalej). Takéto rezanie je potom rekurzívne aplikované na vzniknuté časti až pokým neobsahuje každá časť menej ako určený počet pravidiel. O tom, ktorá dimenzia a na koľko častí sa má rezať, rozhoduje algoritmus na základe vhodne zvolenej heuristiky.

Podobnú stromovú štruktúru a postupné delenie prehľadávaného priestoru využíva aj algoritmus HyperCuts (popísaný napríklad v článku [19]). Tento algoritmus, na rozdiel od algoritmu HiCuts, povoľuje v jednom vrchole delenie priestoru v rámci viacerých dimenzií súčasne.

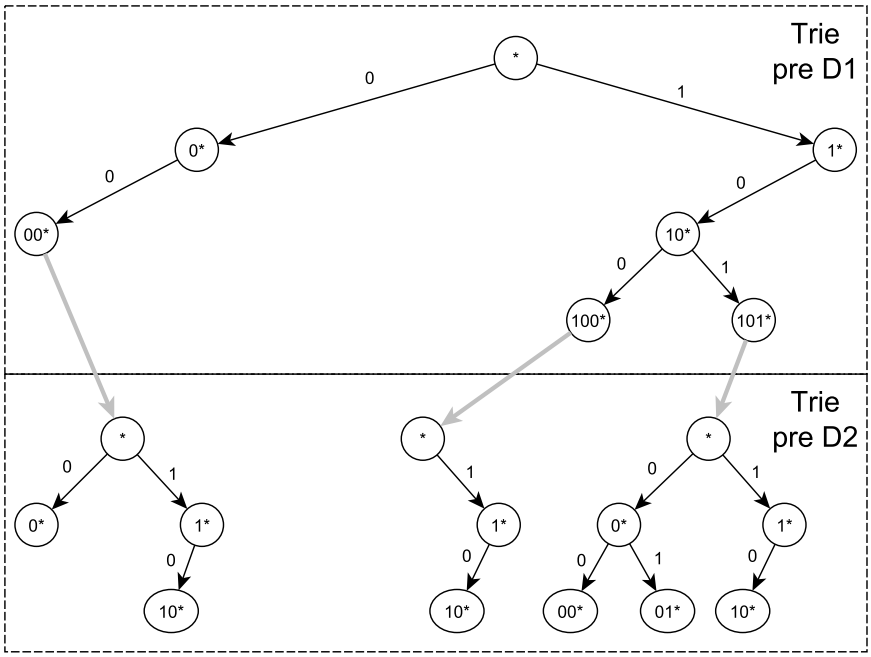


Obr. 5.8: Príklad delenia prehľadávaného priestoru na základe stromovej štruktúri.

Obrázok 5.8 ilustruje princíp delenia (rezania) prehľadávaného priestoru na menšie časti pre jednoduchý binárny strom. Konkrétne ide o ilustráciu rozdelenia dvojrozmerného priestoru na 4 časti pomocou 3 rezov. Pravidlá, ktoré reprezentujú v určitej dimenzii rozsah alebo prefix môžu po rozdelení na časti patriť do viacerých častí (konkrétne na obrázku ide o pravidlo *2*, ktoré patrí do častí *1*, *2* a *4*).

Na rozšírenie trie na vyhľadávanie vo viacerých dimenziách je možné použiť hierarchické trie. Hierarchické trie konštruujeme rekurzívne. Pokiaľ je počet dimenzií *D* väčší ako jedna, zostrojíme trie pre jednu z dimenzií. Pre každý prefix tejto dimenzie potom skonštruujeme hierarchické trie s počtom dimenzií *D-1* nad množinou pravidiel, ktoré vyhovujú danému

prefixu vo zvolenej dimenzii. Príklad hierarchickej trie dátovej štruktúry je vyobrazený na obrázku 5.9.



Obr. 5.9: Príklad stromovej štruktúry hierarchické trie.

Hierarchické trie prinášajú až exponenciálny nárasť priestorovej zložitosti, čo oslabuje možnosť škálovať algoritmus na veľký počet dimenzií a pravidiel.

Hierarchické trie je možné vylepšiť odstránením duplicít v jednotlivých vnorených jednodimenzionálnych trie. V takomto prípade je však nutné spätné vracanie sa, čo zvyšuje časovú zložitosť. Kompromis prináša zavedenie ukazovateľov v rámci jednotlivých úrovní trie, kedy uzly jedného trie môžu ukazovať na ďalšie potencionálne trie rovnakej úrovne.

Pre vyhľadanie vo viacerých dimenziách je všeobecne možné použiť techniku využívajúcu bitové vektory. Pre každú dimenziu a každý unikátny prefix v danej dimenzii je vytvorený bitový vektor, ktorý určuje, v ktorých pravidlách sa daný prefix vyskytuje. Dĺžka bitového vektora je rovná počtu pravidiel. V každej dimenzii potom prebehne samostatne vyhľadanie zodpovedajúceho prefixu a jemu priradeného bitového vektora. Nad nájdenými bitovými vektormi sa vykoná bitový súčin a výsledok určuje vyhľadané pravidlá (výsledný bitový vektor totiž určuje pravidlá, ktoré obsahuju v každej dimenzii nájdenú hodnotu).

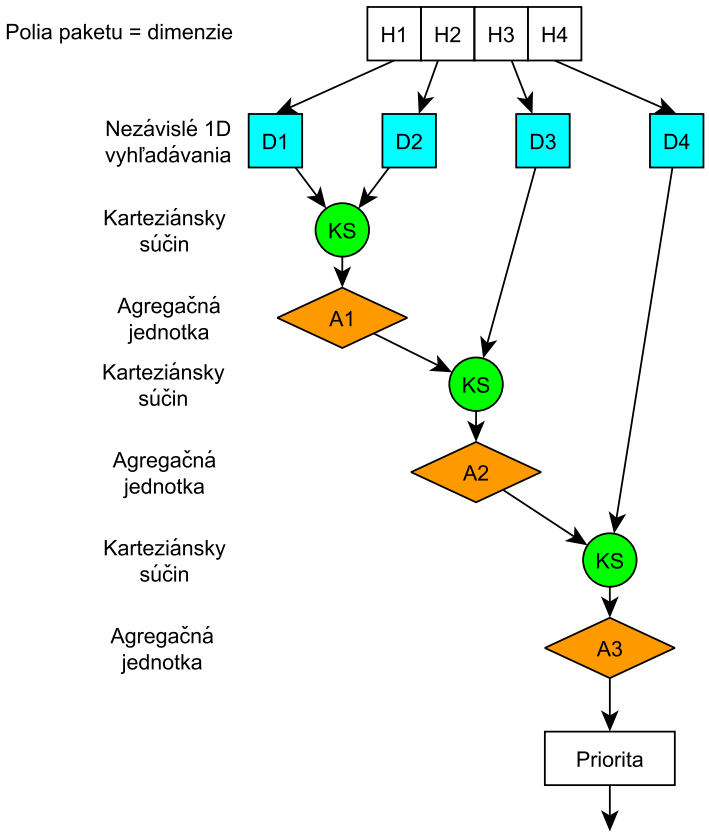
Alternatívou je využitie karteziánskeho súčinu. Nad množinami hodnôt v jednotlivých dimenziách sa spočíta karteziánsky súčin a pre každý prvok tohto súčinu sa určí pravidlo, ktoré danej kombinácii najviac vyhovuje. Pre klasifikáciu paketu potom stačí vyhľadať nezávisle hodnotu v každej dimenzii a spojením týchto hodnôt vzniká prvok karteziánskeho súčinu. Pre tento prvok je už predpočítané pravidlo, ktoré mu vyhovuje a teda netreba nič ďalšie počítať.

Pre veľké množstvo pravidiel však výsledný karteziánsky súčin obsahuje veľký počet prvkov, čo vedie k veľkej pamäťovej zložitosti a neefektívnemu pridávaniu a odoberaniu pravidiel.

Čiastočné zníženie počtu prvkov karteziánskeho súčinu je možné dosiahnuť postupným

vytváraním karteziánskych súčinov len nad dvomi množinami. Môžeme napríklad vytvoriť karteziánsky súčin hodnôt dvoch dimenzií a prvky tohto súčinu rozdeliť do tried ekvivalencie na základe pravidiel, ktoré im vyhovujú (prvky, ktorým vyhovujú rovnaké pravidlá patria do rovnakej triedy ekvivalencie). Rovnaký postup aplikujeme na ďalšie dvojice dimenzií. Tým vytoríme dielčie karteziánske súčiny a rôzne triedy ekvivalencie. Nad triedami ekvivalencie potom môžeme následne opäť po dvojiciach vytvoriť karteziánske súčiny. Takto rekurzívne postupujeme, až kým nemáme výsledné triedy ekvivalencie.

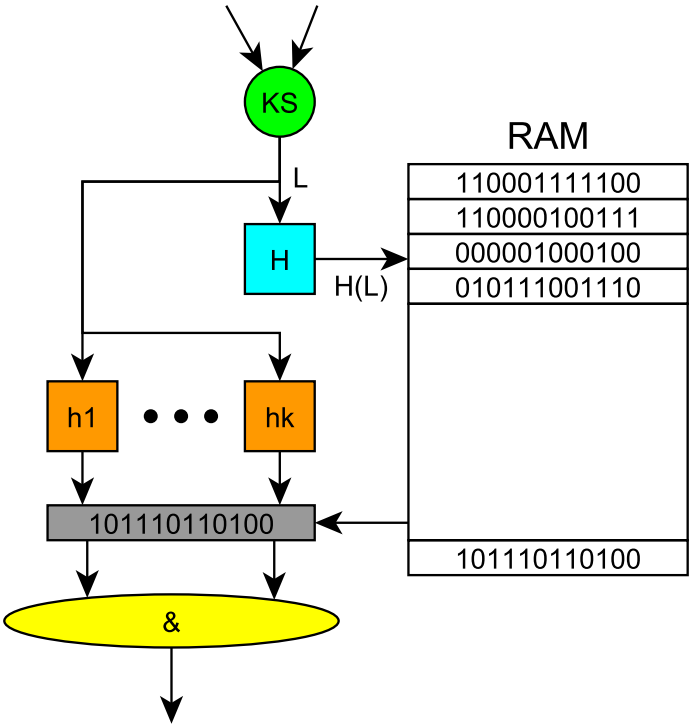
Ternárna pamäť adresovaná obsahom (anglicky ternary content addressable memory, TCAM) je vhodnou harvérovou implementáciou. TCAM dokáže uchovávať jednotlivé hodnoty polí pravidiel spolu s maskou, ktorou je možné reprezentovať prefixy. TCAM navyše poskytuje paralelné porovnanie hľadanej hodnoty so všetkými uloženými hodnotami v jednom takte. Je teda možné v jednom takte vyhľadať všetky pravidlá.



Obr. 5.10: Štruktúra algoritmu DCFL.

Problém takýchto pamätí spočíva v ich náročnosti na zdroje na čipe. Veľkosti TCAM sa pohybujú v jednotkách megabitov, čo nie je pre veľké množiny pravidiel dostačujúce.

Algoritmus DCFL (anglicky distributed crossproducting of field labels, [21]) vychádza zo znalosti, že počet unikátnych hodnôt v jednej dimenzii, s ktorými sa príchádzajúci paket zhoduje, je väčšinou 5 alebo menej. Princíp algoritmu spočíva v rozdelení klasifikácie na jednotlivé dimenzie. V každej dimenzii sa vykoná jedno dimenzionálna klasifikácia, výsledkom ktorej je zoznam vyhovujúcich unikátnych hodnôt pre danú dimenziu. Každá hodnota má priradenú lokálne unikátnu značku (anglicky label). Toto je možné vykonať upravenými



Obr. 5.11: Princíp poľa Bloomových filtrov.

klasickými algoritmami. Následne sa tieto zoznamy spájajú pomocou stromovej štruktúry špeciálnych agregačných uzlov. Túto štruktúru ilustruje obrázok 5.10.

Agregačné uzly vytvárajú karteziánsky súčin spájaných zoznamov. Z predpokladu, že spájané zoznamy obsahujú bežne menej ako 5 prvkov vyplýva, že ani vytvorený karteziánsky súčin nebude obsahovať veľa prvkov. Následne je pre každý prvok súčinu potrebné určiť, či je daná kombinácia obsiahnutá v nejakom pravidle. Ide o problém určenia príslušnosti do množiny, ktorý je možné riešiť napríklad poľom Bloomových filtrov.

Princíp fungovania poľa Bloomových filtrov je ilustrovaný obrázkom 5.11. Bloomov filter počíta k hašovacích funkcií nad hodnotou L. Výsledky hašovacích funkcií určujú bitové pozície v bitovom vektore. Pokiaľ sú hodnoty všetkých týchto bitov nastavené na 1, množina, ktorú Bloomov filter reprezentuje, danú hodnotu obsahuje.

Z dôvodu zníženia počtu prístupov do pamäte sú Bloomove filtre rozšírené na pole Bloomových filtrov. Ide o pole bitových vektorov a hašovaciu funkciu H, ktorá na základe hodnoty L určuje index do tohto poľa. Vybraný bitový vektor je uložený do pomocného registra a nad obsahom registra sa potom vykoná overenie rovnako ako pri Bloomovom filtri.

Problémom tohto prístupu je, že môžu vznikať špeciálne prípady, kedy Bloomov filter považuje niektoré hodnoty za prvky množiny aj v prípade, že prvkami nie sú (anglicky sa tieto prípady nazývaju false-positives). Tento problém sa dá odstrániť pridaním poslednej agregačnej jednotky, ktorá kontroluje príslušnosť do množiny explicitne. Posledná agregačná jednotka teda nemôže byť implementovaná poľom Bloomových filtrov.

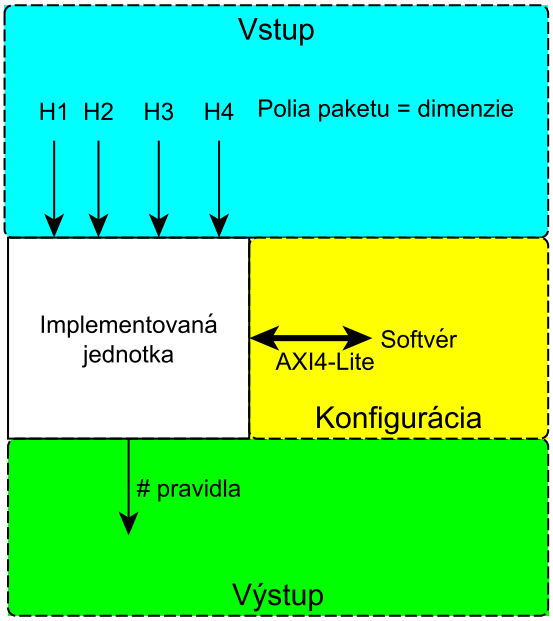
Jednotlivé úrovne agregácie môžu fungovať paralelne, preto je tento algoritmus možné

jednoducho paralelizovať. Algoritmus je teda vhodný hlavne na hardvérovú implementáciu. Oproti iným algoritmom, určeným na klasifikáciu vo viacerých dimenziách, má algoritmus relatívne dobré požiadavky na potrebnú pamäť a poskytuje možnosť pridávania a odoberania pravidiel s náročnosťou ekvivalentnou vyhľadaniu pravidla. Navyše je algoritmus škálovateľný na vyššie počty dimenzií a pravidiel.

Kapitola 6

Návrh a implementácia

Cieľom celej diplomovej práce je zabezpečiť mapovanie vyhľadávacích tabuliek jazyka P4 do FPGA. Preto bola navrhnutá a implementovaná parametrizovateľná hardvérová jednotka. Úlohou tejto jednotky je klasifikovať pakety na základe vyextrahovaného vektoru hodnôt jednotlivých dimenzií. Počet dimenzií a počet pravidiel klasifikácie sa menia v závislosti na konkrétnej vyhľadávacej tabuľke, jednotka je preto dostatočne generická, aby, okrem iného, podporovala rôzne šírky dimenzií a rôzne počty pravidiel. Samotný jazyk P4 nedefinuje konkrétne pravidlá, ale predpokladá, že tieto pravidlá sú pridávané a odoberané počas behu, konkrétne na podnet od riadiaceho softvéru (napríklad podobne ako OpenFlow). Jednotka teda podporuje pridávanie a odoberanie pravidiel počas behu zo softvéru. K jednotlivým vyhľadávacím tabuľkám sa potom, na základe konkrétnych vlastností, generujú rôzne nastavenia tejto hardvérovej jednotky.

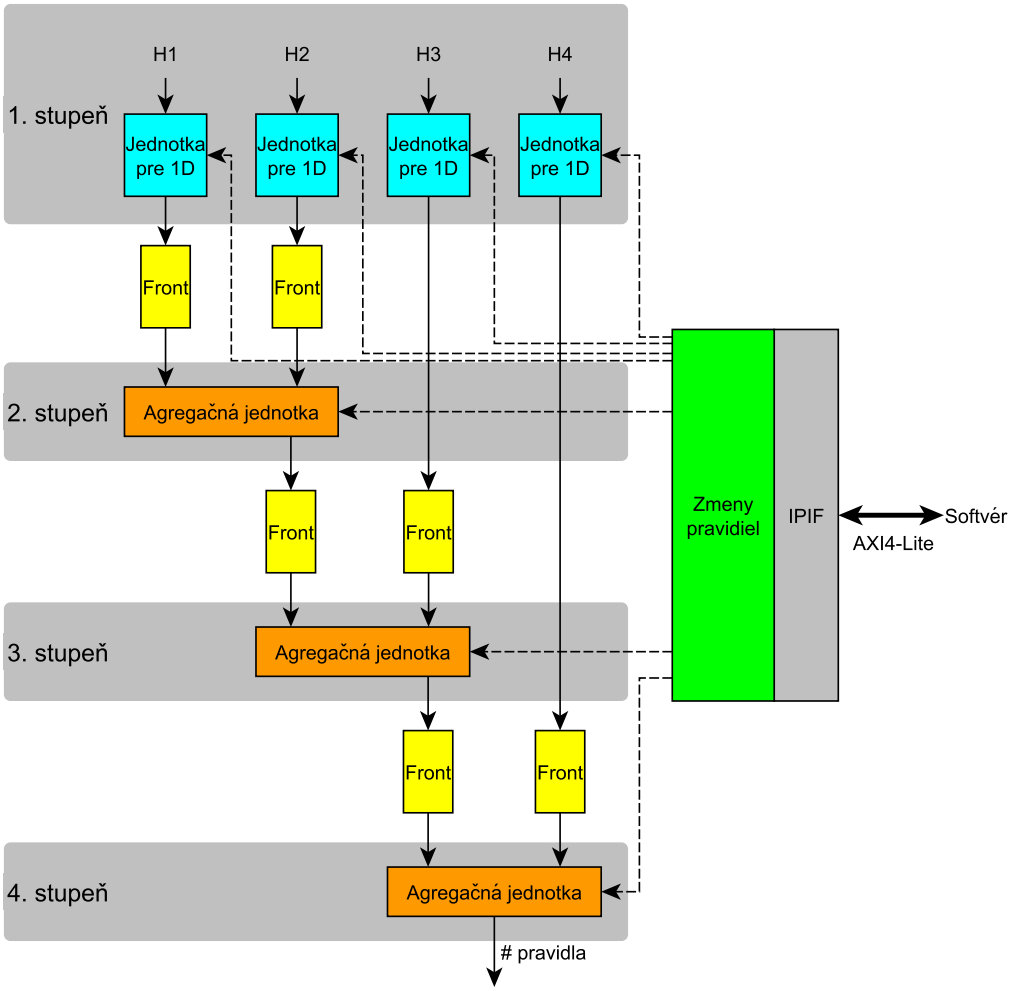


Obr. 6.1: Rozhranie implementovanej hardvérovej jednotky na najvyššej úrovni.

Na najvyššej úrovni abstrakcie jednotka predstavuje implementáciu algoritmu DCFL, ktorý bol, vzhľadom na svoje parametre, zvolený ako najvhodnejší pre účely tejto práce.

Ako už bolo spomenuté v sekcii 5.5, algoritmus DCFL poskytuje dobrú škálovateľnosť, nízku spotrebu zdrojov cieľovej architektúry a vhodnú úroveň paralelizácie.

Obrázok 6.1 zachytáva vstupné, výstupné a konfiguračné rozhranie celej jednotky. Vstupom jednotky na najvyššej úrovni sú jednotlivé hodnoty dimenzií klasifikácie vyextrahované z hlavičky príchodzieho paketu. Výstupom sú čísla pravidiel, ktoré vyhovujú danému paketu. Konfiguračné rozhranie slúži na komunikáciu so softvérom, konkrétne na pridávanie a odoberanie pravidiel počas behu jednotky. Konkrétne ide o AXI4-Lite rozhranie ([24]).



Obr. 6.2: Bloková schéma implementovanej jednotky na najvyššej úrovni abstrakcie.

Bloková schéma na obrázku 6.2 ilustruje najvyššiu úroveň abstrakcie. Klasifikácia prebieha formou zreťazeného spracovania.

Prvý stupeň zreťazeného spracovania predstavujú jednotky na klasifikáciu podľa jednej dimenzie. Každá dimenzia klasifikácie je spracovaná nezávisle (a teda je toto spracovanie možné paralelizovať) jednou z týchto jednotiek. Samotná implementácia týchto jednotiek na spracovanie jednotlivých dimenzií závisí hlavne na tom, aký typ zhody daná dimenzia vyžaduje. Každá jednotka pre klasifikáciu v jednej dimenzii na výstupe dáva zoznam hodnôt, ktoré sú súčasťou nejakého pravidla a zároveň sa zhodujú so vstupnou hodnotou. Z dôvodu šetrenia zdrojov (nižšie bitové šírky signálov) používajú jednotky na svojom výstupe uni-

kátne značky priradené jednotlivým hodnotám v danej dimenzii. Výsledky sa ukladajú do vyrovnávacích frontov, ktoré zabezpečujú synchronizáciu jednotlivých stupňov zreťazeného spracovania.

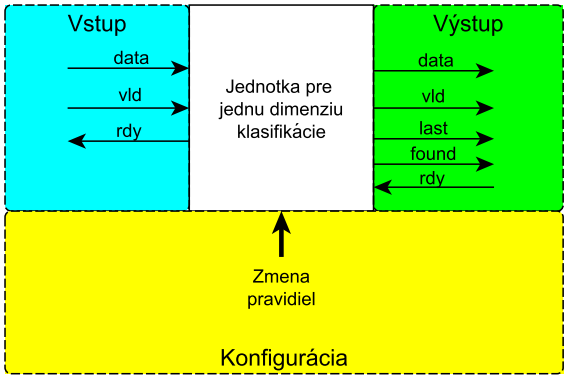
Ďalšie stupne zreťazeného spracovania sú tvorené agregačnými jednotkami, ktoré vždy spájajú 2 zoznamy výsledkov z predchádzajúceho stupňa do jedného zoznamu výsledkov. Agregačná jednotka 2. stupňa, na základe výsledných zoznamov z dvoch dimenzií klasifikácie, vytvorí najprv kartézsky súčin týchto zoznamov a následne pre každý prvok vypočítaného súčinu určí, či je daný prvok súčasťou nejakého pravidla. Pokiaľ je prvok súčasťou pravidla, je vložený na výstup agregačnej jednotky. Výstupom agregačnej jednotky je teda zoznam unikátnych dvojíc. Výstup sa opäť ukladá do vyrovnávajúceho frontu, aby sa zabezpečila synchronizácia s nasledujúcim stupňom zreťazeného spracovania.

Agregačné jednotky ďalších stupňov zreťazeného spracovania potom spájajú výsledný zoznam z agragačnej jednotky predchádzajúceho stupňa s výsledným zoznamom z jednej z ďalších jednotiek na spracovanie jednotlivých dimenzií.

Špeciálna jednotka na spracovanie pridávania pravidiel využíva komponent AXI4-Lite IP Interface (IPIF, [25]) od firmy Xilinx na ovládanie AXI4-Lite rozhrania. Táto jednotka spracuje pridanie (resp. odobranie) pravidla od softvéru a následne jednotlivé zmeny potrebné na pridanie (resp. odobranie) pravidla propaguje ostatným jednotkám.

6.1 Jednotky na spracovanie jednej dimenzie klasifikácie

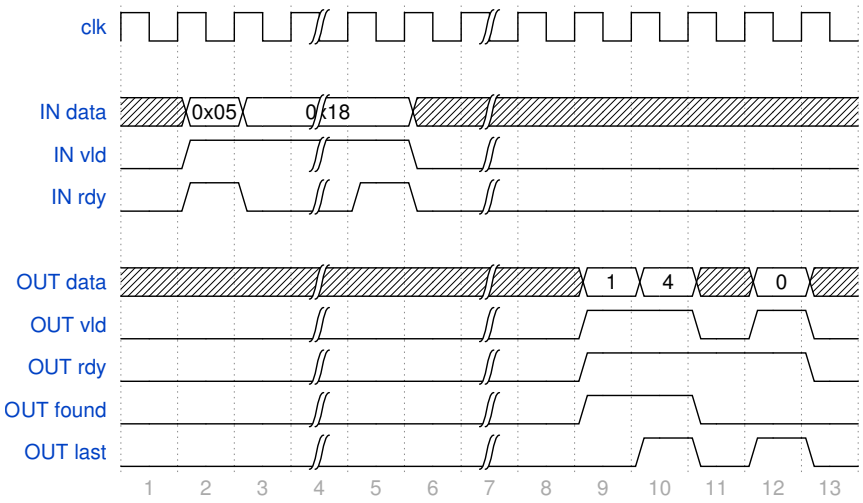
Cieľom týchto jednotiek je spracovanie jednej dimenzie klasifikácie. Kokrétne teda pre vstupnú hodnotu v danej dimenzii vrátiť všetky hodnoty tejto dimenzie, ktoré sú obsiahnuté v nejakom pravidle a zároveň sa s nimi vstupná hodnota zhoduje (s ohľadom na typ zhody).



Obr. 6.3: Rozhranie jednotky na spracovanie jednej dimenzie klasifikácie.

Všeobecné rozhranie jednotky na spracovanie jednej dimenzie klasifikácie popisuje obrázok 6.3. Rozhranie sa dá rozdeliť na tri časti. Vstupná časť rozhrania pozostáva z troch signálov. Signál *data* slúži na prenos hodnoty poľa hlavičky vyextrahovanej z paketu. Platnosť tejto hodnoty signalizuje signál *vld*. Hodnota je však prijatá len v prípade, že je aktívny aj signál *rdy*, ktorý slúži na indikáciu pripravenosti jednotky. Tento signál je aktívny len v prípade, že jednotka momentálne nevykonáva žiadnu inú činnosť.

Výstupná časť rozhrania navyše obsahuje signály *last* a *found*. Signál *last* slúži na oddelenie výsledkov viažucich sa k jednému vstupu. Aktívnosť tohoto signálu znamená, že



Obr. 6.4: Priebeh vstupných a výstupných signálov rozhrania jednotky na spracovanie jednej dimenzie klasifikácie.

momentálne prenášaná hodnota je poslednou patriacou k momentálnemu paketu. Inými slovami tento signál signalizuje koniec zoznamu výsledkov. Signál *found* určuje, či bol vôbec nejaký výsledok nájdený. Priebeh vstupných aj výstupných signálov, pre vyhľadanie hodnoty *0x05*, ktorá vyhovuje pravidlám *1* a *4* a hodnoty *0x18*, ktorá nevyhovuje žiadnemu pravidlu, ukazuje obrázok 6.4. V momente, keď sú oba signály *vld* aj *rdy* vstupného rozhrania aktívne, dochádza k načítaniu vstupnej hodnoty (signál *data*). Určitý čas potom nie je jednotka pripravená prijať ďalšiu vstupnú hodnotu, čo signalizuje neaktívnosťou signálu *rdy*. S určitým oneskorením sa potom na výstupnom rozhraní objavia hodnoty *1* a *4* spolu s aktívnymi signálmi *vld* a *found* výstupného rozhrania. Spolu s hodnotou *4* je ešte navyše aktívny signál *last*. Celkovo táto postupnosť výstupných signálov signalizuje, že pre prvý načítaný vstup sa našla zhoda s pravidlami *1* a *4*. Nakoniec sa na výstupnom rozhraní objaví hodnota *4* spolu s aktívnym signálom *vld* ale neaktívnym singálom *found*, čo znamená, že pre druhý vstup sa nepodarilo nájsť zhodu so žiadnym pravidlom.

Konfiguračná časť rozhrania slúži na pridávanie a odoberanie pravidiel. Konkrétna podoba tejto časti rozhrania závisí na type jednotky. Táto časť obsahuje signály na prenos samotných dát pridávaného pravidla, signál určujúci pripravenosť jednotky pridať pravidlo a prípadne adresu pamäťového miesta, ktoré sa pridaním pravidla prepisuje.

6.1.1 Exaktné porovnanie

Na implementáciu exaktného porovnania z jazyka P4 je vhodné použiť jednotku implementujúcu kukučie hašovanie. Táto jednotka je výhodná hlavne z dôvodu vysokej rýchlosti vyhľadania pri zachovaní nízkej pamäťovej náročnosti. Keďže ide o úplnú zhodu s kľúčom, bude mať výstupný zoznam vždy len jeden prvok. Vstupný signál *last* preto môže byť trvale aktívny. Samotná hardvérová implementácia kukučieho hašovania bola prebratá z projektu Sec6Net[1].

Pre správne vygenerovanie hardvérovej architektúry potrebujeme poznať hlavne nasledujúce tri parametre:

[1]Moderní prostředky pro boj s kybernetickou kriminalitou na Internetu nové generace, dostupné na: http://www.fit.vutbr.cz/research/view_project.php.cs?id=517.

- **Bitová šírka kľúča**. Ide priamo o šírku dimenzie, vieme teda tento parameter získať priamo z popisu vyhľadávacej tabuľky v jazyku P4.
- **Bitová šírka dát, ktoré sa pomocou kukučieho hašovania ukladajú**. Tieto dáta predstavujú unikátne značky priradené jednotlivým hodnotám dimenzie (teda kľúčom). Potrebná bitová šírka preto závisí na maximálnom počte unikátnych značiek v danej dimenzii. Počet týchto značiek je určite zhora obmedzený maximálnou kapacitou vyhľadávacej tabuľky.
- **Počet položiek v jednotlivých tabuľkách adresovaných kukučím hašovaním**. Táto hodnota sa dá opäť odvodiť z maximálnej kapacity vyhľadávacej tabuľky a počtu tabuliek, ktoré kukučie hašovanie využíva.

Niektoré ďalšie nastaviteľné parametre kukučieho hašovania nie je možné určiť priamo z popisu v jazyku P4. Preto sú tieto parametre nastavované na implicitné hodnoty, prípadne ich môže užívateľ definovať v prídavnom konfiguračnom súbore. Týmito parametrami sú:

- **Počet tabuliek**. Ide o počet tabuliek a zároveň hašovacích funkcií, ktoré kukučie hašovanie využíva. Aby fungoval algoritmus efektívne odporúča sa použiť hodnoty *2*, *3* alebo *4*. Pre vyššie hodnoty dochádza k neefektívnemu využívaniu pamäte.
- **Typ blokových pamätí**. Tento parameter určuje, koľko-bitové blokové pamäte sa majú využiť na implementáciu tabuliek kukučieho hašovania. Pripustné sú 1-bitové, 2-bitové, 4-bitové, 9-bitové, 18-bitové a 36-bitové pamäte.

6.1.2 Bitová maska, prefixy a rozsahy

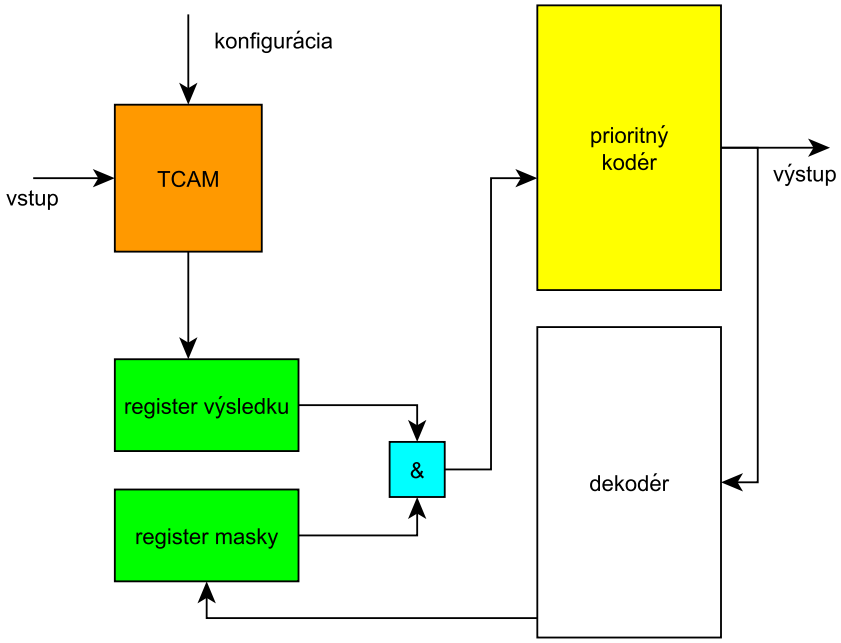
Porovnanie na inú ako exaktnú zhodu vyžaduje využitie iných prístupov ako kukučie hašovanie. Na implementáciu zhody s využitím bitovej masky je z dôvodu vysokej rýchlosti vhodné použiť pamäť TCAM. Pamäť TCAM je možné priamo použiť aj pre zhodu na prefix. Pri zhode typu rozsah je potrebné rozsah previesť na množinu prefixov, ktoré je už možné riešiť priamo pamäťou TCAM. Samotná hardvérová implementácia pamäte TCAM bola prebratá z projektu SProbe[2].

Jednotku na spracovanie jednej dimenzie klasifikácie využívajúcu pamäť TCAM zobrazuje obrázok 6.5. Pamäť TCAM vracia všetky výsledky paralelne v jednom takte v podobe bitového vektora. Tento vektor je potrebné previesť na postupnosť výsledkov, ktorá je kompatibilná s rozhraniami ostatných jednotiek. Na to slúži pomocná logika, ktorá v každom takte pomocou prioritného kodéru určí najvyšší bit vektora, ktorý je aktívny. Tento bit následne vymaskuje a v ďalšom takte pracuje s vymaskovaným vektorom na zistenie nasledujúceho výsledku.

Pre správne vygenerovanie hardvérovej architektúry jednotky potrebujeme poznať hlavne nasledujúce parametre:

- **Bitová šírka položky pamäti**. Tento parameter predstavuje bitovú šírku dimenzie.
- **Kapacita pamäti**. Maximálny počet položiek pamäti musí byť aspoň tak veľký, ako počet unikátnych značiek v tejto dimenzii. Rovnako ako pri kukučom hašovaní je počet unikátnych značiek v dimenzii zhora obmedzený kapacitou vyhľadávacej tabuľky.

[2]Sondy pro analýzu a filtraci provozu na úrovni aplikačních protokolů, dostupné na: http://www.fit.vutbr.cz/research/view_project.php.cs?id=1003.



Obr. 6.5: Jednotka na spracovanie jednej dimenzie klasifikácie využívajúca TCAM.

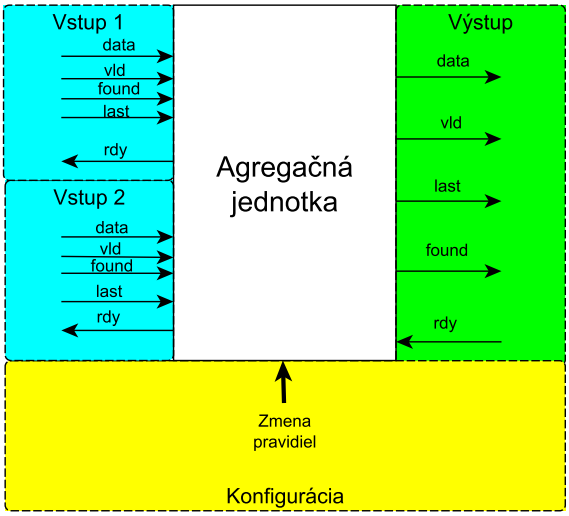
Ďalšie nastaviteľné parametre jednotky využívajúcej TCAM, ktoré však nejde určiť priamo z popisu vyhľadávacej tabuľky v jazyku P4 a sú nastavované na implicitné hodnoty alebo ich užívateľ nastavuje pomocou prídavného konfiguračného súboru, sú:

- **Faktor sekvenčnosti**. Faktor sekvenčnosti (f_s) určuje pomer medzi rýchlosťou vyhľadania položky v TCAM a množstvom zdrojov, ktoré TCAM zaberá. Môže nadobúdať hodnotu celého čísla v rozsahu od *0* do *6*. Jedna vyhľadávacia tabuľka (anglicky lookup table, skrátene LUT) so šiestimi vstupmi na FPGA čipe vie uložiť 2^{f_s} položiek TCAM šírky $6-f_s$ bitov. Čas vyhľadania je potom 2^{f_s} cyklov. Vyššia hodnota faktoru sekvenčnosti znamená menej zabraných zdrojov, ale zároveň nižšiu priepustnosť.
- **Prednosť zápisu pred vyhľadaním**. Je potrebné určiť, či má vyššiu prioritu zápis novej hodnoty alebo vyhľadanie.

6.2 Agregačné jednotky

Úlohou agregačných jednotiek je na základe dvoch vstupných zoznamov vytvoriť zoznam výstupný, pričom výstupný zoznam obsahuje len prvky kartézskeho súčinu vstupných zoznamov, ktoré sú obsiahnuté v nejakom z pravidiel. Agregačné jednotky teda postupne spájajú výsledky z jednotlivých dimenzií, až sa nakoniec dostanú k celkovému výsledku klasifikácie.

Všeobecné rozhranie agregačných jednotiek ilustruje obrázok 6.6. Rozhranie sa skladá zo štyroch rôznych častí. Na rozdiel od jednotiek na spracovanie jednotlivých dimenzií klasifikácie majú agregačné jednotky dve vstupné časti rozhrania, z oboch príjmajú zoznamy výsledkov z predchádzajúcich úrovní zreťazeného spracovania. Protokol je rovnaký ako pri jednotkách na spracovanie jednotlivých dimenzií, teda hodnota je načítaná s nábežnou



Obr. 6.6: Rozhranie agregačnej jednotky.

hranou hodinového signálu len v prípade, že sú signály *rdy* a *vld* aktívne a koniec jedného zoznamu výsledkov je prenášaný pomocou signálu *last*.

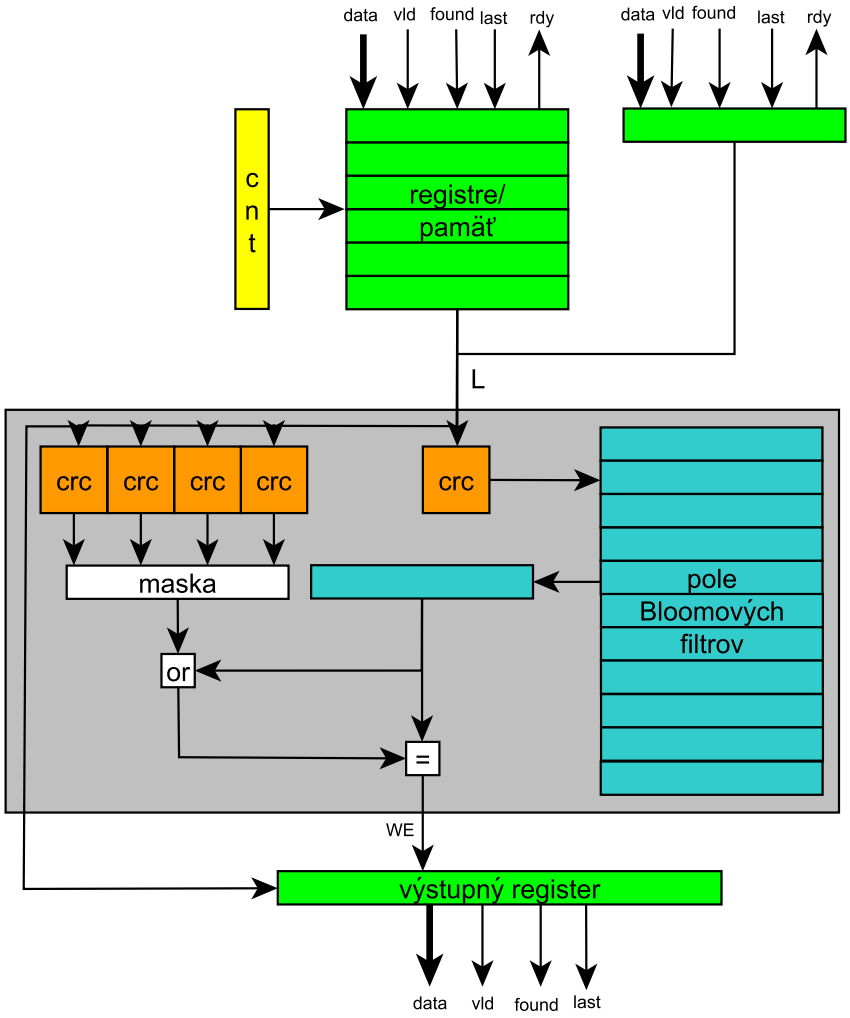
6.2.1 Využitie poľa Bloomových filtrov

Na implementáciu agregačnej jednotky je veľmi efektívne použiť pole Bloomových filtrov. Keďže ide o pravdepodobnostnú štruktúru, poskytuje vhodný kompromis medzi potrebnou pamäťou, rýchlosťou a správnosťou vyhľadania. Samotná implementácia poľa Bloomových filtrov je rozšírená o riadiacu logiku, ktorá sa stará o vytvorenie kartézkeho súčinu vstupných zoznamov, o správne nastavenie výstupných signálov a rieši špeciálne prípady, ako napríklad prípad, keď je jeden zo vstupných zoznamov prázdny, alebo prípad, keď žiadna z dvojíc kartézskeho súčinu nepatrí do žiadneho z pravidiel.

Architektúru jednotky približuje obrázok 6.7. Jednotka začína prácu načítaním jedného zo zoznamov. Následne sa postupne načítajú prvky druhého zoznamu, pričom pre každý prvok sa pomocou jednoduchého počítadla prejdú všetky prvky prvého (už načítaného) zoznamu a privedú sa takto vytvorené dvojice na vstup komponentu implementujúceho pole Bloomových filtrov. V rámci tohoto komponentu sú nad privedenými dvojicami vypočítané hašovacie funkcie (konkrétne CRC s rôznymi inicializačnými hodnotami). Výsledok jednej z hašovacích funkcií určuje adresu do pamäti, z ktorej sa načíta do registeru obsah jedného Bloomového filtra. Výsledky ostatných hašovacích funkcií sa použijú na nastavenie bitov masky. Dvojica je obsiahnutá v nejakom pravidle pokiaľ Bloomov filter obsahuje jednotky na všetkých pozíciách, na ktorých obsahuje jednotky aj maska. Toto je jednoducho overiteľné porovnaním Bloomového filtra s bitovým súčtom Bloomového filtra a masky.

Agregačná jednotka je implementovaná genericky, konkrétne je možné nastaviť tieto parametre:

- **Bitová šírka prvého vstupu**. Predstavuje bitovú šírku potrebnú na unikátne označenie možných výsledkov v prvom zozname.
- **Bitová šírka druhého vstupu**. Predstavuje bitovú šírku potrebnú na unikátne označenie možných výsledkov v druhom zozname.



Obr. 6.7: Agregačná jednotka využívajúca pole Bloomových filtrov.

- **Počet hašovacích funkcií**. Parameter (k) určuje koľko hašovacích funkcií sa využije. Väčší počet použitých hašovacích funkcií znamená menšiu pravdepodobnosť výskytu false-positive. Konkrétne sa pravdepodobnosť výskytu false-positive (p) riadi vzťahom:

$$p = \left(\frac{1}{2}\right)^k \tag{6.1}$$

  Aby však bolo pole Bloomových filtrov schopné uložiť N položiek pri zachovaní pravdepodobnosti výskytu false-positive z rovnice 6.1, musí obsahovať aspoň F Bloomových filtrov. Hodnota F sa vypočíta nasledujúcim vzťahom:

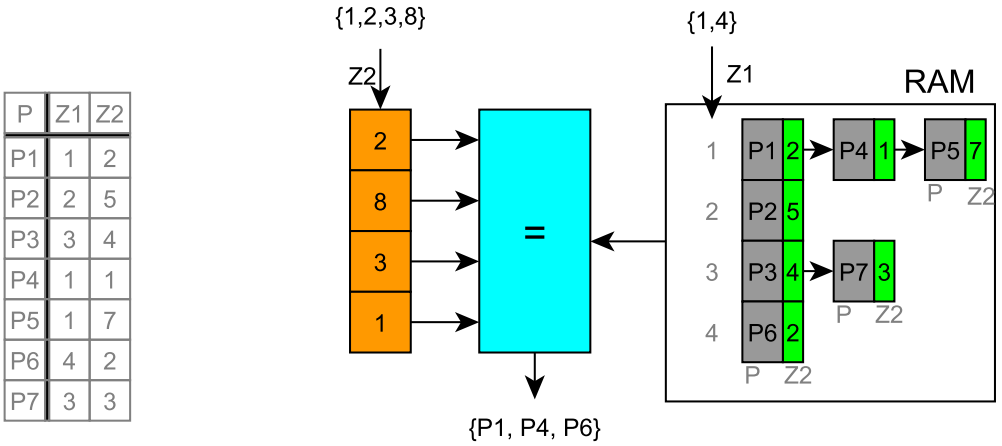
$$F = \frac{k * N}{32 * ln(2)} \approx N * \frac{k}{22} \tag{6.2}$$

- **Počet Bloomových filtrov**. Tento parameter určuje koľko 32-bitových Bloomových filtrov sa bude využívať. Aby bola zachovaná pravdepodobnosť false-positive z rovnice

6.1, mal by byť tento počet nastavený podľa rovnice 6.2.

6.2.2 Indexovanie pomocou značiek

Využitie Bloomových filtrov, ktoré predstavujú pravdepodobnostnú štruktúru, prináša problém s výskytom false-positive. Ide o problém, kedy môžu byť niektoré vstupné kombinácie nájdené aj v prípade, že tieto kombinácie nie sú súčasťou žiadneho pravidla. V prvých stupňoch zreťazeného spracovania je následkom len potenciálne zvýšenie počtu výsledkov, ktoré vstupujú do nasledujúce stupňa. Reálne sa toto zvýšenie pohybuje v jednotkách percent, čo je z pohľadu rýchlosti zanedbateľné. Väčší vplyv má tento problém v poslednom stupni zreťazeného spracovania, kedy môže viesť na nesprávny výsledok celej klasifikácie. Preto nie je možné v poslednom stupni zreťazeného spracovania využívať agregačnú jednotku založenú na Bloomových filtroch ani na žiadnej inej pravdepodobnostnej štruktúre.

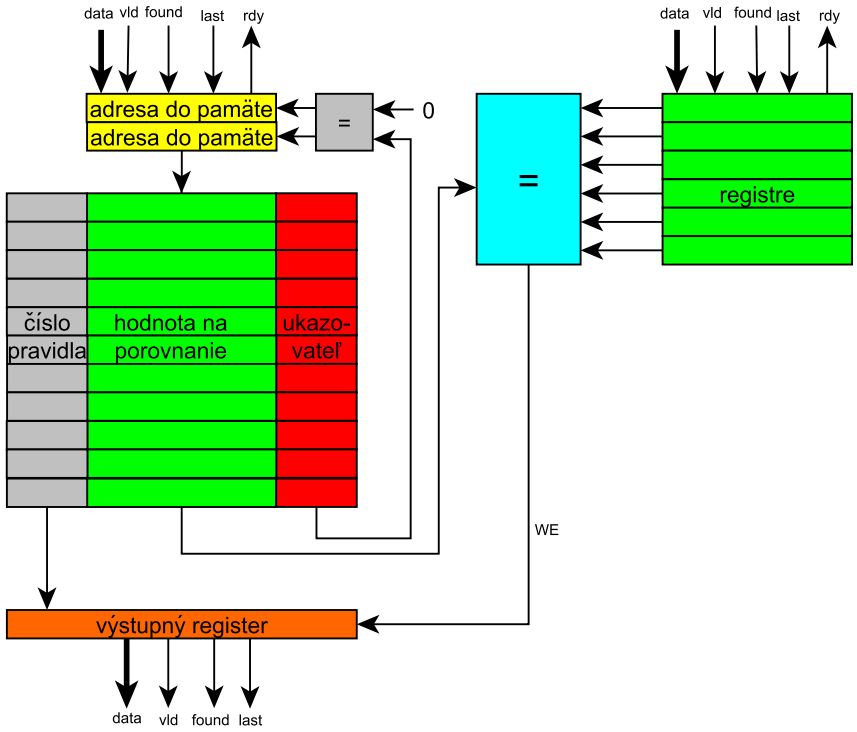


Obr. 6.8: Ilustrácia princípu indexovania pomocou značiek.

Navyše je vhodné, aby v poslednom stupni zreťazeného zpracovania bolo (okrem určenia, či je vyšetrovaná n-tica, obsahujúca unikátnu značku z každej dimenzie, obsiahnutá v nejakom pravidle) určené, ktoré pravidlo vyšetrovanú n-ticu obsahuje. Ide teda o prevod n-tice značiek jednotlivých dimenzií na číslo pravidla, ktoré tejto n-tici zodpovedá. Aby sme splnili vyššie uvedené požiadavky, musíme explicitne ukladať n-ticu značiek spolu s číslom pravidla.

Technika indexovania pomocou značiek je znázornená na obrázku 6.8. Táto technika je založená na tom, že v pamäti ukladáme pre každú unikátnu značku jedného zo vstupov agregačnej jednotky spájaný zoznam. Každý prvok spájaného zoznamu obsahuje nasledujúce 3 položky:

- **Číslo pravidla**. Táto položka obsahuje číslo nejakého pravidla, ktoré v sebe obsahuje danú značku (to znamená značku, ktorej spájaný zoznam prísluší).
- **Unikátne značky v ostatných dimenziách**. Táto položka predstavuje pravidlo zapísané ako n-ticu unikátnych značiek všetkých dimenzií.
- **Ukazovateľ na ďalší prvok**. Keďže sa jedná o spájaný zoznam, musí každý prvok obsahovať ukazovateľ na svojho následníka.



Obr. 6.9: Architektúra jednotky používajúcej indexovanie pomocou značiek.

Každý spájaný zoznam, patriaci k nejakej unikátnej značke, teda reprezentuje všetky pravidlá, v ktorých je táto značka obsiahnutá.

V pamäti teda udržujeme pole začiatkov takýchto spájaných zoznamov. Značku z jedného zo vstupov agregačnej jednotky potom môžeme využiť ako index do tohoto poľa. Zodpovedajúci spájaný zoznam postupne prechádzame a každý prvok zoznamu porovnávame s celým zoznamom hodnôt prijatým na druhom vstupe agregačnej jednotky. Pri nájdení zhody vieme, že konkrétna dvojica vstupov, pri ktorej zhoda nastala v skutočnosti predstavuje pravidlo.

Celkovú štruktúru agregačnej jednotky využívajúcej techniku indexovania pomocou značiek zachytáva obrázok 6.9. Podobne ako agregačná jednotka využívajúca Bloomové filtre aj táto jednotka najprv musí načítať celý zoznam z jedného zo vstupov. Následne načítava postupne zoznam z druhého vstupu, pričom načítané hodnoty používa ako adresu do pamäti. Spájaný zoznam na danej adrese je postupne prechádzaný a na harvérovej úrovni paralelne porovnávaný zo všetkými vopred načítanými prvkami zoznamu z prvého vstupu. Počet paralelne vykonaných porovnaní je limitovaný na 15, čo je však z pohľadu reálnych množín pravidiel dostačujúce (bližšie rozobrané v ďalšej kapitole). Koniec spájaného zoznamu indikuje ukazovateľ ukazujúci na adresu *0*.

Z dôvodu oneskorenia pri čítaní dát z blokovej pamäti, nie je možné prechádzať jeden spájaný zoznam efektívne. Časť čítaných dát totiž predstavuje adresu nasledujúceho prvku spájaného zoznamu, preto je vždy potrebné počkať 1 takt, kým vieme, z akej adresy potrebujeme čítať v ďalšom kroku. Táto neefektívnosť je vyriešená striedaním spracovania dvoch rôznych spájaných zoznamov. Teda v takte, v ktorom spracovanie jedného spája-

ného zoznamu musí čakať na dáta z blokovej pamäti, môže pokračovať spracovanie druhého spájaného zoznamu, pre ktorý už boli potrebné dáta z pamäti načítané.

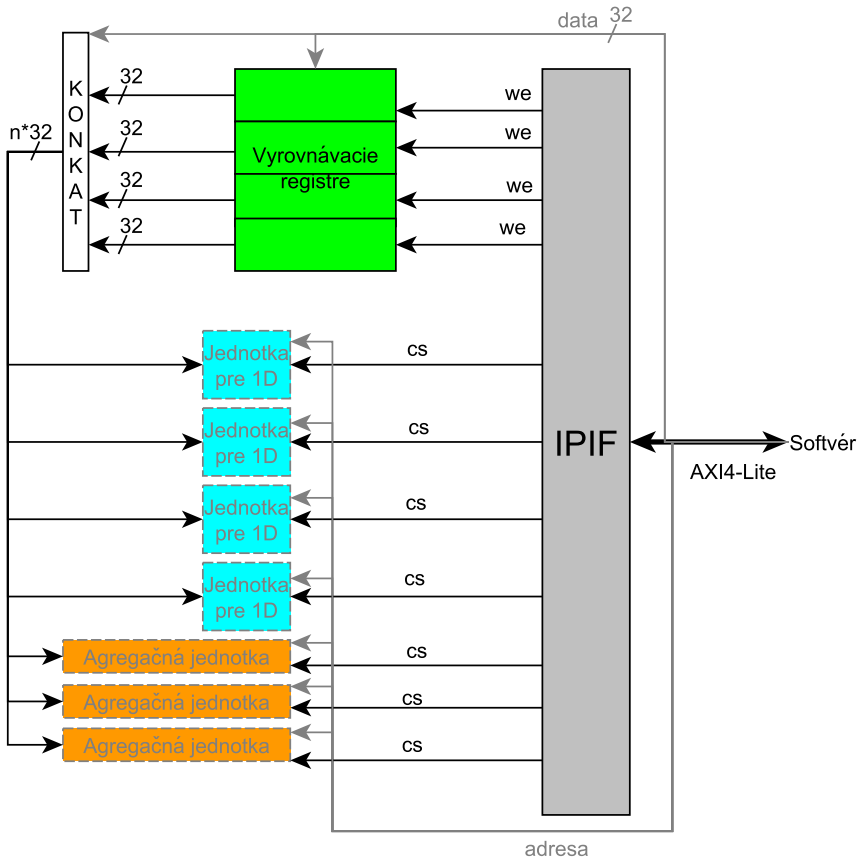
Dôležité parametre sú, rovnako ako pri jednotke využívajúcej Bloomové filtre, najmä bitové šírky vstupných značiek (W_1, W_2) a počet položiek pamäti, pričom pre počet položiek pamäti M pri maximálnom počte pravidiel N by malo platiť:

$$2^{W_1} \leq M \leq N \tag{6.3}$$

Pokiaľ nerovnica 6.3 neplatí, môžu nastať dva problémy. Buď existuje nejaká značka, ktorou nie je možné pamäť adresovať, alebo existujú pamäťové miesta, ktoré nebudú nikdy (ani v najhoršom možnom prípade) využité.

6.3 Nahrávanie pravidiel

Úlohou tohoto komponentu je zabezpečiť správne pridávanie a odoberanie pravidiel. Pridávanie a odoberanie pravidiel je riadené zo softvéru, a preto tento komponent musí byť schopný komunikovať so softvérom.



Obr. 6.10: Architektúra jednotky na spracovanie zmien pravidiel.

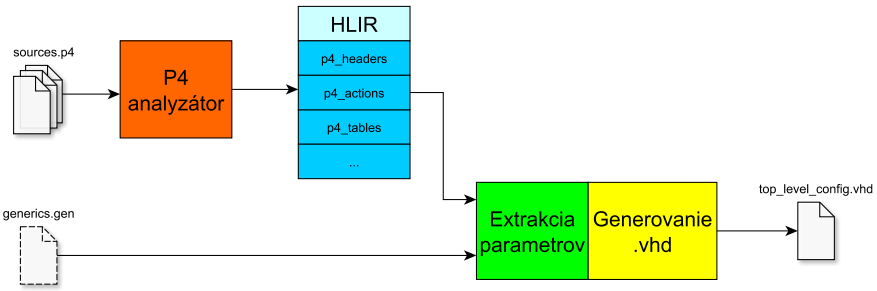
Obrázok 6.10 ilustruje štruktúru komponentu. Na komunikáciu so softvérom slúži rozhranie AXI4-Lite ([24]), ktoré je z hardvérovej strany riadené komponentom IPIF ([25])

od firmy Xilinx, ktorý okrem zabezpečenia správnej komunikácie na rozhraní AXI4-Lite, poskytuje aj adresový dekodér. IPIF teda nastavuje *cs* (chip select) signály, ktoré povoľujú prácu jednotlivých vnútorných registrov a pamätí v jednotlivých komponentoch.

AXI4-Lite rozhranie má fixnú šírku dátového signálu 32 bitov, jednotlivé pamäti však môžu mať šírku položiek väčšiu ako 32 bitov. Preto v prípade, že pridanie pravidla vyžaduje zápis do nejakej z týchto pamätí, sú dáta presahujúce 32 bitov prenesené vo viacerých taktoch a ukladajú sa najprv do vyrovnávacích registrov. Až po prenesení posledných 32 bitov dát je povolený zápis do pamäti, pričom dátový signál zbernice AXI4-Lite je konkatenovaný s výstupmi vyrovnávacích registrov. Vyrovnávacie registre sú využité aj v prípade, že pridanie pravidla do niektorej z jednotiek vyžaduje viac ako 32 bitov dát. Napríklad pridanie hodnoty IP adresy do pamäte TCAM, vyžaduje okrem samotnej hodnoty poznať v momente zápisu aj masku. Preto sa najprv zapíše maska do vyrovnávacieho registra a až následne sa vykoná samotný zápis hodnoty do pamäťového priestoru pamäte TCAM.

Samotná rýchlosť nahrávania pravidiel závisí na počte dimenzií, ich šírke a type zhody v každej z týchto dimenzií. Napríklad predpokladajme 3 32-bitové dimenzie so shodou typu prefix implementované pomocou pamätí TCAM, pričom maximálny počet zapísaných pravidiel je 128. Na pridanie jedného pravidla potrebujeme zapísať 64 bitov pre každú dimenziu (32b hodnota v danej dimenzii a 32b maska danej dimenzie), 14 bitov pre pole Bloomových filtrov implementujúce prvú agregačnú jednotku (2-krát 7b pre unikátnu značku z jednotlivých dimenzií) a 56 bitov pre jednotku indexovania značkou implementujúcu druhú agragačnú jednotku (7b unikátny identifikátor pravidla, 14b explicitne uložená kombinácia prvých dvoch dimenzií, 7b ukazovateľ na nasledujúci prvok spájaného zonamu, v najhoršom prípade musíme upraviť dva prvky spájaného zoznamu). Celkovo pre tento prípad potrebujeme preniesť 392 bitov, na čo potrebujeme aspoň 13 cyklov. Za predpokladu, že AXI4-Lite pracuje na frekvencii 200 MHz to zodpovedá pridaniu približne 15 miliónov pravidiel za sekundu.

Vďaka využitiu vyrovnávacích registrov je zabezpečené, že pridanie pravidla prebehne v jednom kroku a nemôže dochádzať k nezrovnalostiam v pamätiach jednotlivých jednotiek. Je preto možné pridávať pravidlá bez nutnosti zastaviť klasifikáciu.



Obr. 6.11: Ilustrácia mapovania vyhľadávacích tabuliek jazyka P4 na nastavenie hardvérovej jednotky.

6.4 Mapovanie z jazyka P4

Vyhľadávacie tabuľky jazyka P4 je možné jednoducho mapovať na vyššie popísanú architektúru. Proces mapovania znázorňuje obrázok 6.11.

Jednotlivé dôležité generické parametre komponentov už boli popísané a väčšinu týchto parametrov je možné získať priamo zo súboru s popisom v jazyku P4, prípadne ich z tohoto popisu odvodiť. Prvým krokom mapovania je preto spracovanie a analýza zdrojových P4 súborov. Túto úlohu vykoná P4 analyzátor, ktorý momentálne funguje ako knižnica jazyka Python. Tento analyzátor prevádza zdrojové súbory do vnútornej reprezentácie (anglicky high-level intermediate representation, skrátene HLIR). Táto reprezentácia predstavuje hierarchiu rôznych objektov, ktoré zodpovedajú jednotlivým entitám jazyka P4.

Z HLIR sú následne extrahované parametre, ktoré sú z pohľadu mapovania zaujímavé. Ide hlavne o parametre týkajúce sa jednotlivých vyhľadávacích tabuliek, ktoré boli v zdrojových súboroch obsiahnuté. Na základe získaných parametrov sú nastavené všetky odvoditeľné parametre. Parametre, ktoré nie je možné získať odvodením z popisu v jazyku P4 sú zatiaľ nastavené na implicitné hodnoty. Mapovanie okrem zdrojového súboru obsahujúceho popis jazyka P4 načíta aj rozširujúci konfiguračný súbor (ak takýto súbor existuje) a na jeho základe zmení niektoré implicitné hodnoty.

Nakoniec je vygenerovaný VHDL súbor, obsahujúci balíček jednotlivých generických parametrov zapísanych formou konštánt. Hodnoty týchto konštánt sú vypočítané zo získaných hodnôt parametrov. Súčasťou výsledného balíčka je aj popis pamäťového priestoru, ktorý je taktiež odvodený z ostatných parametrov.

Samotná hardvérová jednotka je na najvyššej úrovni implementovaná genericky (s využitím konštrukcií *generate* jazyka VHDL), kedy vygenerovanie jednotlivých komponentov je podmienené hodnotami konštánt, ktoré sú obsiahnuté v konfiguračnom balíčku. Pre syntézu architektúr pre rôzne parametre teda stačí iba zmeniť obsah konfiguračného balíčka (resp. nahradiť tento balíček vygenerovaným balíčkom).

Kapitola 7

Analýza výsledkov

Množiny pravidiel využité na analýzu kvality implementovaného algoritmu boli prevzaté z nástroja NetBench ([18]). Ide väčšinou o množiny pravidiel nad piatimi dimenziami, konkrétne zdrojová IP adresa, cieľová IP adresa, zdrojový port, cieľový port, protokol, predstavujúcimi identifikátor sieťového toku. V tabuľkách v tejto kapitole sú dimenzie často označené skratkami *zIP*, *cIP*, *zp*, *cp*, *p* zodpovedajúcimi postupne zdrojovej IP adrese, cieľovej IP adrese, zdrojovému portu, cieľovému portu a protokolu. V dimenziách zdrojovej a cieľovej IP adresy obsahujú pravidlá typ zhody s prefixom. Dimenzie zdrojového a cieľového portu predstavujú typ zhody na rozsah a dimenzia protokolu predstavuje typ úplnej zhody.

7.1 Charakteristiky množín pravidiel

Základnou charakteristikou každej množiny pravidiel je ich počet. Túto charakteristiku pre

Množina pravidiel	**Počet pravidiel**		
	Originál	**Bez duplicít**	**Rozgenerované rozsahy**
fw1_05_05	822	733	733
fw1_05_m05	882	830	830
fw1_m05_05	827	717	717
fw1_m05_m05	852	773	773
fw2_05_05	956	941	941
fw3_05_05_500	472	451	451
fw4_05_05_500	482	477	477
fw5_05_05_500	481	459	459
acl1	753	740	2406
acl1_100	99	97	97
acl1_1K	917	889	889
fw1	270	259	259
fw1_100	93	87	87
fw1_1K	792	726	726
ipc1	1550	1485	1550
ipc1_100	100	100	100
ipc1_1K	938	927	974

Tabuľka 7.1: Počet pravidiel v jednotlivých množinách pravidiel.

jednotlivé množiny obsahuje tabuľka 7.1. Okrem samotného počtu orignálnych pravidiel obsahuje tabuľka aj hodnoty po odstránení duplicít, teda po odstránení dvojíc rovnakých pravidiel, prípadne pravidiel, ktoré sa navzájom pokrývajú, pričom pokryté pravidlo má nižšiu prioritu. Navyše je možné vidieť aj hodnoty po rozgenerovaní rozsahov na prefixy. Je vhodné si všimnúť, že všetky množiny pravidiel obsahujú rádovo rovnaký počet pravidiel.

Množina pravidiel	**Počet unikátnych hodnôt**							
	zIP	**cIP**	**zp**	**cp**	**p**	**zIP+cIP**	**zIP+zp**	**zIP+cp**
fw1_05_05	138	123	13	43	1	329	230	298
fw1_05_m05	157	171	13	43	1	363	264	340
fw1_m05_05	133	146	13	43	1	357	221	327
fw1_m05_m05	122	143	13	41	1	326	223	320
fw2_05_05	735	274	9	0	1	908	802	0
fw3_05_05_500	103	52	9	39	1	295	156	293
fw4_05_05_500	54	82	27	46	1	288	179	279
fw5_05_05_500	105	84	11	32	1	319	172	278
acl1	97	205	0	140	1	426	0	466
acl1_100	34	62	0	39	1	85	0	82
acl1_1K	103	297	0	99	1	493	0	619
fw1	57	66	13	43	1	128	87	112
fw1_100	12	25	11	26	1	37	23	39
fw1_1K	124	175	13	42	1	317	208	219
ipc1	152	128	33	53	1	941	216	530
ipc1_100	63	63	14	14	1	93	73	76
ipc1_1K	336	436	27	44	1	843	455	592

Tabuľka 7.2: Počet unikátnych hodnôt v jednotlivých dimenziách.

Z hladiska algoritmu DCFL, ktorý postupne vyhodnocuje jednotlivé dimenzie a následne ich kombinácie, je ešte dôležitou charakteristikou počet unikátnych hodnôt v jednotlivých dimenziách. Tento údaj zachytáva tabuľka 7.2. Ďalšou zaujímavou charakteristikou je počet unikátnych kombinácií hodnôt v dvoch dimenziách. Informáciu o počte týchto kombinácií obsahuje taktiež tabuľka 7.2. Je celkom jasne vidieť, že počet unikátnych hodnôt v jednej dimenzii je niekoľkonásobne nižší ako počet pravidiel. Postupným spájaním dimenzií sa však táto hodnota približuje počtu pravidiel. Podrobnejšie výsledky je možné nájsť v prílohe v tabuľkách B.1, B.2 a B.3.

Zaujímavou charakteristikou množiny pravidiel z pohľadu algoritmu DCFL je aj počet prefixov jednej dimenzie, ktoré sú v sebe vnorené (teda jeden prefix je prefixom druhého). Maximálny a priemerný počet návzajom v sebe vnorených prefixov pre jednotlivé dimenzie je v tabuľke 7.3. Tento počet sa, až na jednu výnimku, pohybuje v rádoch jednotiek, čo je veľmi dôležité pre efektívne fungovanie algoritmu DCFL. Je možné si všimnúť, že priemerné hodnoty sú podľa očakávaní o niečo nižšie ako maximálne hodnoty. Medzi týmito hodnotami a počtom pravidiel neexistuje veľmi silná závislosť. Vzájomné vnorenie pravidiel závisí hlavne na štruktúre jednotlivých pravidiel danej množiny. Výsledky pre rôzne kobinácie dimenzií je možné nájsť v prílohe v tabuľkách B.4, B.5, B.6, B.7, B.8 a B.9.

Množina pravidiel	Maximálne vnorenie					Priemerné vnorenie				
	zIP	cIP	zp	cp	p	zIP	cIP	zp	cp	p
fw1_05_05	4	4	3	3	1	3	3	2	2	1
fw1_05_m05	4	5	3	3	1	3	3	2	2	1
fw1_m05_05	4	4	3	3	1	3	3	2	2	1
fw1_m05_m05	4	5	3	3	1	3	4	2	2	1
fw2_05_05	3	3	2	0	1	3	2	1	0	1
fw3_05_05_500	4	3	3	3	1	3	3	2	2	1
fw4_05_05_500	3	4	4	3	1	3	3	3	2	1
fw5_05_05_500	5	4	3	3	1	5	4	2	2	1
acl1	4	4	0	54	1	3	2	0	46	1
acl1_100	4	5	0	3	1	3	2	0	2	1
acl1_1K	4	4	0	5	1	3	3	0	3	1
fw1	4	4	3	3	1	2	2	2	2	1
fw1_100	4	2	3	3	1	3	1	2	2	1
fw1_1K	4	4	3	3	1	3	2	2	2	1
ipc1	4	5	3	5	1	2	3	2	3	1
ipc1_100	4	5	3	3	1	2	2	2	2	1
ipc1_1K	4	5	3	5	1	3	3	2	3	1

Tabuľka 7.3: Maximálne a priemerné vnorenie jednotlivých prefixov v rámci dimenzií.

7.2 Priepustnosť

Na zistenie priepustnosti výslednej implementácie potrebujeme najprv analyzovať ďalšie charakteristiky množín pravidiel. Zaujímajú nás najmä počty výsledkov, ktoré môžu jednotlivé úrovne algoritmu DCFL vracať pre jeden paket. Z tabuliek 7.4 a 7.5 je možné vidieť, že maximálny a priemerný počet kombinácií hodnôt dimenzií, ktorým môže jeden paket vyhovovať (teda inými slovami maximálny počet výsledkov úrovne zreťazeného spracovania algoritmu DCFL pre jeden paket) je závislý na poradí, v ktorom sú dimenzie spracované.

Poradie dimenzií	Maximálny počet kombinácií			Maximálna veľkosť kart. súčinu		
	1D+2D	1D+2D+3D	1D+2D+3D+4D	1D+2D	1D+2D+3D	1D+2D+3D+4D
zIP,cIP,zp,cp,p	8	14	10	16	24	42
zIP,zp,cIP,cp,p	10	14	10	12	40	42
zIP,cp,zp,cIP,p	9	10	10	12	27	40
cIP,cp,zIP,zp,p	10	14	10	12	40	42
cIP,zp,zIP,cp,p	10	14	10	12	40	42
cIP,zp,p,cIP,cp	10	10	14	12	10	40

Tabuľka 7.4: Maximálny počet kombinácií hodnôt dimenzií, ktorým môže vyhovovať jeden paket a maximálna veľkosť kartézskeho súčinu pre rôzne permutácie poradia dimenzií pre množinu pravidiel *fw1_05_05*.

Z týchto hodnôt je možné navyše spočítať maximálnu a priemernú veľkosť kartézskeho súčinu, ktorý musí byť jednou úrovňou agregačných jednotiek spracovaný. Veľkosti týchto kartézskych súčinov sú taktiež závislé na poradí spracovania dimenzií a ich hodnoty sú v tabuľkách 7.4 a 7.5.

Samotná priepustnosť pre určité poradie spracovania dimenzií je potom závislá na naj-

Poradie dimenzií	Priemerný počet kombinácií			Priemerná veľkosť kart. súčinu		
	1D+2D	1D+2D+3D	1D+2D+3D+4D	1D+2D	1D+2D+3D	1D+2D+3D+4D
zIP,cIP,zp,cp,p	5	6	5	9	10	12
zIP,zp,cIP,cp,p	5	6	5	6	15	12
zIP,cp,zp,cIP,p	4	4	5	6	8	12
cIP,cp,zIP,zp,p	5	6	5	6	15	12
cIP,zp,zIP,cp,p	4	6	5	6	12	12
cIP,zp,p,cIP,cp	4	4	6	6	4	12

Tabuľka 7.5: Priemerný počet kombinácií hodnôt dimenzií, ktorým môže vyhovovať jeden paket, priemerná veľkosť kartézskeho súčinu pre rôzne permutácie poradia dimenzií pre množinu pravidiel *fw1_05_05*.

Množina pravidiel	Najhorší prípad			Priemerne		
	Cyklov na paket	Priepustnosť		Cyklov na paket	Priepustnosť	
		Mpkt/s	Gb/s		Mpkt/s	Gb/s
fw1_05_05	40	5	2.56	12	16.66	8.52
fw1_05_m05	33	6.06	3.1	10	20	10.24
fw1_m05_05	36	5.56	2.84	10	20	10.24
fw1_m05_m05	36	5.56	2.84	12	16.66	8.54
fw2_05_05	12	16.66	8.54	6	33.33	17.06
fw3_05_05_500	27	7.4	3.8	12	16.66	8.54
fw4_05_05_500	48	4.16	2.14	18	11.12	5.68
fw5_05_05_500	39	5.12	2.62	20	10	5.12
acl1	148	1.36	0.7	46	4.34	2.24
acl1_100	20	10	5.12	6	33.33	17.06
acl1_1K	28	7.14	3.66	12	16.66	8.54
fw1	20	10	5.12	4	50	25.6
fw1_100	14	14.28	7.32	4	50	25.6
fw1_1K	28	7.14	3.66	9	22.22	11.38
ipc1	25	8	4.1	9	22.22	11.38
ipc1_100	21	9.52	4.88	4	50	25.6
ipc1_1K	32	6.26	3.2	12	16.66	8.54

Tabuľka 7.6: Najhoršie a priemerné hodnoty priepustnosti pre jednotlivé množiny pravidiel pre algoritmus DCFL a pre frekvenciu hodín 200 MHz a veľkosť paketov 64B.

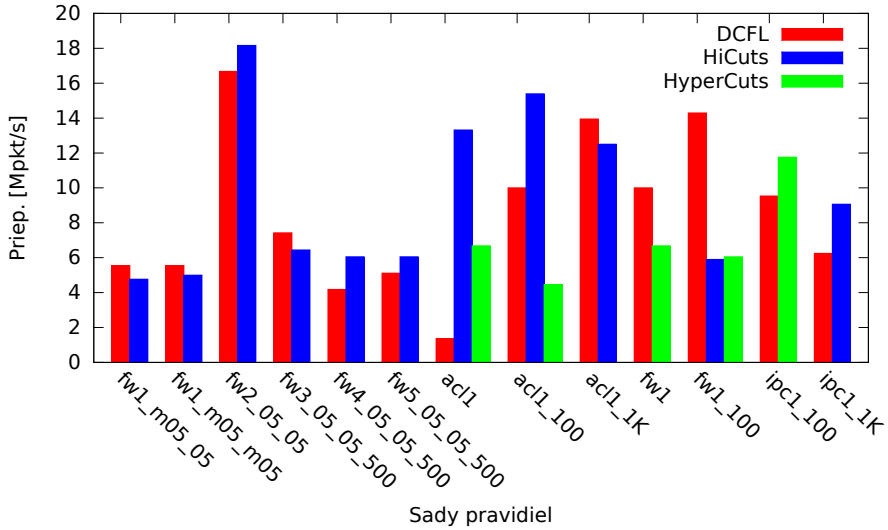
väčšom kartézskom súčine pre toto poradie. Pre množinu pravidiel *fw1_05_05* a poradie spracovania dimenzií *(zdroj. IP adresa, zdroj. port, cieľ. IP adresa, cieľ. port, protokol)* (2. riadok tabuľky 7.4) má najväčší kartézsky súčin *42* prvkov, zatiaľčo pre poradie spracovania dimenzií *(zdroj. IP adresa, cieľ. port, zdroj. port, cieľ. IP adresa, protokol)* (3. riadok tabuľky 7.4) má len *40* prvkov. Z hľadiska priemerných hodnôt je rozdiel ešte väčší, konkrétne *15* verzus *12* prvkov.

Tabuľka 7.6 obsahuje pre každú množinu pravidiel najhoršiu a priemernú priepustnosť pre najlepšie možné poradie spracovania dimenzií. Ide konkrétne o priepustnosť v miliónoch paketov za sekundu a v Gb za sekundu pre frekvenciu 200 MHz, ktorú je riešenie schopné dosiahnuť a prípad, keď by všetky prichádzajúce pakety mali minimálnu možnú dĺžku 64B. Pre množinu *fw1_05_05* je najlepšie poradie spracovania dimenzií napríklad *(zdroj. IP*

adresa, zdroj. port, cieľ. port, cieľ. IP adresa, protokol). Pre toto poradie je na vyhľadanie jedného paketu potrebných maximálne *40* cyklov, čo zodpovedá vyhľadaniu *2,5* milióna paketov za sekundu, čo v prípade 64B paketov znamená priepustnosť *1,28* Gb za sekundu.

Množina pravidiel	**DCFL[Mpkt/s]**	**HiCuts[Mpkt/s]**	**HyperCuts[Mpkt/s]**
fw1_m05_05	5.56	4.76	-
fw1_m05_m05	5.56	5	-
fw2_05_05	16.66	18.18	-
fw3_05_05_500	7.4	6.46	-
fw4_05_05_500	4.16	6.06	-
fw5_05_05_500	5.12	6.06	-
acl1	1.36	13.33	6.67
acl1_100	10	15.38	4.44
acl1_1K	7.14	12.5	-
fw1	10	-	6.66
fw1_100	14.28	5.88	6.06
ipc1_100	9.52	-	11.76
ipc1_1K	6.26	9.1	-
priemerne	7.92	9.34	7.12

Tabuľka 7.7: Priepustnosť algoritmu DCFL pre najhorší prípad oproti algoritmom HiCuts a HyperCuts pre najhorší prípad vyhľadania.



Obr. 7.1: Graf porovnávajúci priepustnosti pre najhorší prípad implementovaného DCFL oproti algoritmom HiCuts a HyperCuts.

Graf na obrázku 7.1 a tabuľka 7.7 obsahujú porovnanie priepustnosti pre najhoršie prípady vyhľadania algoritmu DCFL oproti algoritmom HiCuts a HyperCuts, ktorých priepustnosti boli odhadnuté pomocou nástroja NetBench.

Je možné vidieť, že implementovaný algoritmus DCFL sa, až na niektoré výnimky, pohybuje okolo rýchlosti algoritmov HiCuts a HyperCuts. Veľkou výhodou implementovaného algoritmu DCFL je však možnosť jednoduchého zvyšovania stupňa paralelnosti, kedy sa dá

algoritmus jednoducho upraviť tak, že sa zvýši priepustnosť za cenu spotrebovaných zdrojov a pamäti. Navyše algoritmy HiCuts a HyperCuts neposkytujú efektívne pridávanie a odoberanie pravidiel počas behu. Po pridaní alebo odobraní pravidla by sa totiž musela prepočítať celá stromová štruktúra týchto algoritmov, čo je výpočtovo náročné.

7.3 Pamäťové nároky

Analýzu pamäťových nárokov je možné rozdeliť na dve časti. Prvou je analýza pamäťových nárokov v prípade, keď bližšie nepoznáme štruktúru pravidiel množiny. V takomto prípade musíme predpokladať, že počty unikátnych hodnôt v jednotlivých dimenziách sú v najhoršom prípade rovnaké ako samotný počet pravidiel. Pre tento prípad obsahuje závislosť potrebnej pamäti pre jednotlivé súčasti implementácie (Bloomove filtre, indexovanie značkou a jednotky na spracovanie jednej dimenzie), ako aj pre celú implementáciu, tabuľka 7.8. Prvé tri stĺpce tabuľky predstavujú parametre klasifikácie, konkrétne postupne

# prav.	# dim.	šírka dim.[b]	BF[B]	IZ[B]	1D[B]	DCFL[B]
128	3	80	96	448	1920	2464
128	3	48	96	448	1152	1696
128	4	80	192	560	1920	2672
128	4	96	192	560	2304	3056
128	6	80	384	784	1920	3088
128	6	128	384	784	3072	4240
256	3	80	188	1024	3840	5052
256	3	48	188	1024	2304	3516
256	4	80	376	1280	3840	5496
256	4	96	376	1280	4608	6264
256	6	80	752	1792	3840	6384
256	6	128	752	1792	6144	8688
1024	3	80	740	5120	15360	21220
1024	3	48	740	5120	9216	15076
1024	4	80	1480	6400	6144	23240
1024	4	96	1480	6400	18432	26312
1024	6	80	2960	8960	6144	27280
1024	6	128	2960	8960	24576	36496

Tabuľka 7.8: Potrebná pamäť pre bližšie neurčenú množinu pravidiel v závislosti na počte dimenzií, ich šírke a počte pravidiel.

počet pravidiel, počet dimenzií a celkovú bitovú šírku dimenzií. Ďalšie stĺpce potom obsahujú minimálne potrebné množstvo pamäti pre jednotky implementujúce pole Bloomových filtrov, indexovanie značkou a vyhľadanie v jednej dimenzii. Nakoniec v poslednom stĺpci sú celkové pamäťové nároky celej implementovanej jednotky. V tabuľke sú pamäťové požiadavky jednotiek na spracovanie jednotlivých dimenzií zastúpené odhadom pamäťových požiadaviek pre kukučie hašovanie. Z tabuľky je možné vyvodiť, že pamäťové požiadavky rastú približne lineárne so zvyšujúcim sa počtom pravidiel, počtom dimenzií a celkovou šírkou jednotlivých dimenzií. Konkrétnejšie je možné približné pamäťové nároky odhadnúť pomocou vzťahu 7.1. V tomto vzťahu majú premenné nasledujúci význam. Premenná m je

množstvo celkovej pamäti (v bajtoch), m_{bfa} je množstvo pamäti spotrebovanej poľami Bloomových filtrov (v bajtoch), m_{li} je množstvo pamäti spotrebovanej jednotkou indexovania pomocou značiek (v bajtoch), m_{1D} je odhad množstva pamäti spotrebovanej jednotkami na vyhľadanie v jednej dimenzii (v bajtoch), d je počet dimenzií, p je počet pravidiel a w je celková šírka dimenzií (v bitoch).

$$m = m_{bfa} + m_{li} + m_{1D} = 4 * (d-2) * \left\lceil \frac{4 * n}{32 * ln(2)} \right\rceil + \frac{n * (d+1) * \lceil log_2(n) \rceil}{8} + \frac{3 * n}{2} * \frac{w}{8} \quad (7.1)$$

Množina pravidiel	**BF[B]**	**IZ[B]**	**1D[B]**	**DCFL[B]**
fw1_05_05	964	4490	1734	7188
fw1_05_m05	1428	5395	2136	8959
fw1_m05_05	1100	4661	1842	7604
fw1_m05_m05	1300	4928	1152	7380
fw2_05_05	121	6352	6081	13645
fw3_05_05_500	864	2424	1074	4362
fw4_05_05_500	832	2683	1035	4550
fw5_05_05_500	884	2639	1263	4786
acl1	388	4718	2232	7338
acl1_100	132	461	693	1286
acl1_1K	1088	2697	6001	9786
fw1	412	1457	906	2775
fw1_100	128	359	333	820
fw1_1K	1112	4628	1959	7699
ipc1	1504	9467	1938	12909
ipc1_100	208	475	840	1523
ipc1_1K	1800	6489	4845	13134

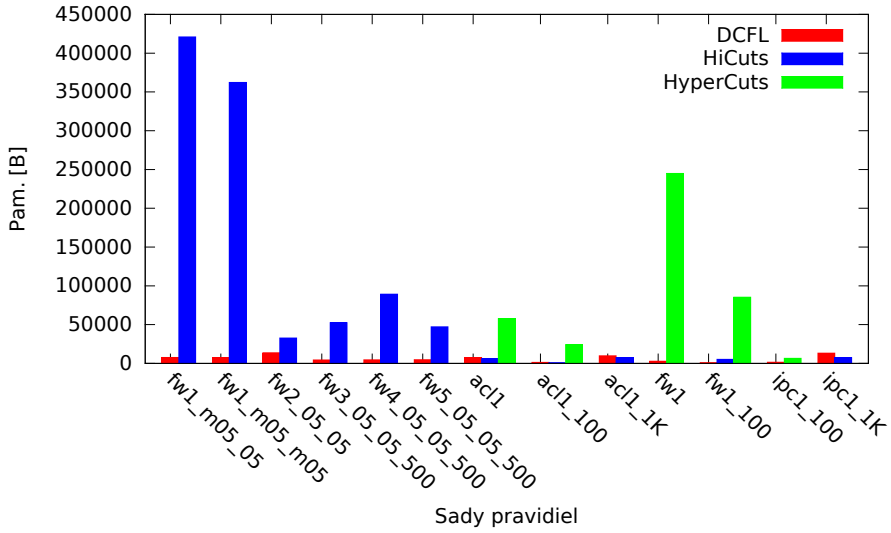
Tabuľka 7.9: Odhad potrebnej pamäti pre konkrétne množiny pravidiel.

Druhou časťou analýzy pamäťových nárokov je odhad týchto nárokov pre konkrétne množiny pravidiel. V tomto prípade sa môže vhodne použiť štruktúra pravidiel na zníženie potrebnej pamäti. Najviac k zníženiu potrebnej pamäti prispieva fakt, že počet unikátnych hodnôt v jednotlivých dimenziách, ktoré treba explicitne ukladať, môže byť niekoľkonásobne menší ako počet pravidiel. Celková pamäť vyhradená pre jednotky na vyhľadanie v jednej dimenzii preto tiež klesá. Odhad množstva potrebnej pamäti pre jednotlivé množiny pravidiel je zhrnutý v tabuľke 7.9. Tabuľka okrem výsledkov pre celé riešenie obsahuje aj výsledky pre jednotlivé časti riešenia (pole Bloomových filtrov, indexovanie značkou jednotky na vyhľadanie v jednej dimenzii). Hodnoty pre algoritmus DCFL nezohľadňujú fakt, že v samotnom hadvéri algoritmus využíva niekoľko blokových pamätí paralelne a niektoré z týchto blokových pamätí nemusia byť plne využité. DCFL preto reálne spotrebuje o niečo viac pamäti.

Tabuľka 7.10 a graf na obrázku 7.2 potom obsahujú porovnanie získaných hodnôt spotreby pamäti s pamäťou potrebnou algoritmami HiCuts a HyperCuts a percentuálnu úsporu pamäti algoritmu DCFL oproti lepšej z variánt HiCuts a HyperCuts. Záporné hodnoty úspory pamäti indikujú, že algoritmus DCFL vyžaduje viac pamäti ako lepší z algoritmov HiCuts a HyperCuts. Z tejto tabuľky a grafu je vidieť, že implementovaný algoritmus DCFL potrebuje často rádovo len desatinu pamäti. Tento jav je spôsobený jednak tým, že algoritmus DCFL využíva na zníženie pamäťovej náročnosti už spomínaný fakt, že počet

Množina pravidiel	DCFL[B]	HiCuts[B]	HyperCuts[B]	Úspora DCFL[%]
fw1_m05_05	7604	420933	-	98
fw1_m05_m05	7380	362193	-	98
fw2_05_05	13645	32645	-	58
fw3_05_05_500	4362	52541	-	92
fw4_05_05_500	4550	89205	-	95
fw5_05_05_500	4786	47113	-	90
acl1	7338	6173	57764	-19
acl1_100	1286	773	24216	-66
acl1_1K	9786	7525	-	-30
fw1	2775	-	244864	99
fw1_100	820	5233	85372	85
ipc1_100	1523	-	6568	77
ipc1_1K	13134	7265	-	-81
priemerne	6076	93782	83757	46

Tabuľka 7.10: Odhady potrebnej pamäti algoritmov HiCuts a HyperCuts a porovnanie s implementovaným algoritmom DCFL.



Obr. 7.2: Graf porovnávajúci pamäťové nároky algoritmov HiCuts a HyperCuts oproti algoritmu DCFL.

unikátnych hodnôt v jednotlivých dimenziách je nižší ako počet pravidiel. Navyše algoritmy HiCuts a HyperCuts využívajú stromovú štruktúru, ktorej veľkosť (teda počet uzlov stromu) často, hlavne pre veľké a komplexné množiny pravidiel, rastie rýchlejšie ako lineárne.

7.4 Spotreba zdrojov na čipe

V neposlednom rade je veľmi dobrou metrikou kvality implementovaného riešenia aj spotreba zdrojov na čipe. Spotreby pre rôzne konfigurácie sa nachádzajú v tabuľke 7.11. Ide konkrétne o spotrebu zdrojov na čipe xc7k160t z rodiny Kintex 7 ([26]). Hodnoty boli

# prav.	# dim.	šír. dim.[b]	typ 1D	frek. [MHz]	reg.	LUT	BRAM
128	3	80	TCAM	274	1526	2210	2
128	3	48	TCAM	274	1397	1991	2
128	4	96	TCAM	274	1866	2614	3
128	6	128	TCAM	270	3270	4417	5
256	4	96	TCAM	239	4121	7212	3
512	4	96	TCAM	236	7234	12296	3
1024	4	96	TCAM	226	13430	23178	4
1024	4	96	Kuk. haš.	347	1459	1814	19
2048	4	96	Kuk. haš.	361	1541	1866	22
4096	4	96	Kuk. haš.	238	1623	2133	38
8192	4	96	Kuk. haš.	300	1758	2321	89
16384	4	96	Kuk. haš.	332	1834	2362	187
32768	4	96	Kuk. haš.	330	1910	2389	394

Tabuľka 7.11: Spotrebované zdroje na čipe xc7k160t z rodiny Kintex 7.

získané pomocou syntézy v programe Xilinx ISE Design Suite 14.1. Prvé štyri stĺpce tabuľky reprezentujú parametre hardvérovej architektúry. Ide postupne o maximálny počet pravidiel, počet dimenzií, celkovú bitovú šírku dimenzií a typ implementácie jednotiek spracovania jednej dimenzie. Piaty stĺpec obsahuje maximálnu frekvenciu, na ktorej je riešenie schopné pracovať. Posledné tri stĺpce potom obsahujú samotnú spotrebu zdrojov, konkrétne počty registrov, LUT a blokových pamätí.

Z tabuľky je jasne vidieť, že samotné agregačné jednotky, jednotka pridávania pravidiel a prepojenia týchto jednotiek zaberajú relatívne málo zdrojov. Najnáročnejšie sú jednotky pre vyhľadávanie v jednej dimenzii. Jedným extrémom sú pamäti TCAM, ktoré vyžadujú veľké množstvo LUT (pre *1024* pravidiel a *4* dimenzie je to až pätina všetkých dostupných LUT). Druhým extrémom je kukučie hašovanie, ktoré vyžaduje hlavne veľa blokových pamätí.

Kapitola 8

Záver

Cieľom tejto diplomovej práce bolo mapovanie vyhľadávacích tabuliek jazyka P4 do technológie FPGA. Vyhľadávacie tabuľky jazyka P4 predstavujú nástroj na klasifikáciu paketov, bolo preto potrebné navrhnúť a implementovať hardvérovú jednotku, schopnú klasifikovať pakety. Následne ešte bolo potrebné navrhnúť samotný spôsob mapovania. Tento cieľ bol v rámci dimplomovej práce splnený.

Riešenie som začal naštudovaním a rozborom potrebných teoretických poznatkov. Ide najmä o princípy klasifikácie paketov a samotného novovznikajúceho jazyka P4, ktorý predstavuje abstraktný nástroj na popis spracovania paketov na sieťovej úrovni. Výsledky štúdijnej fázy sú zhrnuté v kapitolách 2 až 5.

Na základe naštudovaných princípov som následne vytvoril návrh hardvérovej jednotky schopnej klasifikovať pakety pomocou algoritmu DCFL. Začal som návrhom na najvyššom stupni abstrakcie, keď som potrebnú funkcionalitu jednotky rozdelil do viacerých nezávislých blokov. Potom som postupne prechádzal na nižšie úrovne abstrakcie a navrhol som vnútornú štruktúru jednotlivých blokov. Navrhnutú jednotku som implementoval a otestoval. Samotná hardvérová jednotka je parametrizovateľná. Je ju možné použiť na klasifikáciu podľa rôzneho počtu rôzne širokých dimenzií. Jednotka navyše podporuje pridávanie a odoberanie pravidiel počas behu. Nakoniec som ešte vytvoril jednoduché mapovanie vyhľadávacích tabuliek jazyka P4 na konfiguráciu implementovanej jednotky. Popis výsledného návrhu a implementácia sa nachádza v kapitole 6. Výsledok tejto diplomovej práce teda poskytuje prepojenie medzi jazykom P4 a architektúrou FPGA.

Využitím algoritmu DCFL som dosiahol veľmi nízke požiadavky na pamäť. Implementovaná jednotka vyžaduje podstatne menej pamäti ako algoritmy HiCuts a HyperCuts, niekedy až viac ako 10-násobne menej. Súčasne má navrhnutá architektúra porovnateľnú výkonnosť. Pomer priepustností implementovanej jednotky a algoritmov HiCuts a HyperCuts sa, až na niektoré výnimky, pohybuje v rozmedzí dvoch až štyroch tretín. Pri pracovnej frekvencii 200 MHz je možné dosiahnuť zpracovanie na wire-speed pre 10 Gb linky.

Implementovanú jednotku je možné ďalej paralelizovať, čím je možné dosiahnuť ešte vyšších priepustností za cenu zvýšenia pamäťových nárokov. Jednotka navyše pozostáva z blokov, ktoré je možné ďalej nezávisle optimalizovať a vylepšovať. Hlavne pri blokoch, ktoré slúžia na klasifikáciu v jednej dimenzii, je možné použiť takmer ľubovoľný optimalizovaný algoritmus (napríklad rôzne optimalizované trie štruktúry).

Výsledné riešenie je schopné na čipe FPGA z rodiny Kintex 7 pracovať v závislosti na použitých implementáciách jednotiek na spracovanie jednej dimenzie s množinami obsahujúcimi až desiatky tisíc pravidiel. Zároveň zaberá rádovo len jednotky percent logiky dostupnej na FPGA čipe.

Problematikou rozobranou v tejto práci sa plánujem ďalej zaoberať v rámci doktorandského štúdia a dizertačnej práce. Cieľom je dosiahnuť vyššie priepustnosti (pohybujúce sa rádovo v stovkách Gb/s), pri zachovaní čo najnižších pamäťových nárokov. Navyše je plánované implementovanú jednotku nasadiť v rámci projektu SProbe, kde bude slúžiť ako filter paketov pre sondu monitorujúcu sieťovú komunikáciu.

Literatúra

[1] Bosshart, P.; Daly, D.; Gibb, G.; aj.: P4: Programming Protocol-Independent Packet Processors. *ACM SIGCOMM Computer Communication Review*, ročník 44, č. 3, 2014: s. 88–95.
URL http://www.sigcomm.org/sites/default/files/ccr/papers/2014/July/0000000-0000004.pdf

[2] Braden, R.: Requirements for Internet Hosts - Communication Layers. RFC 1122, Október 1989.
URL http://www.ietf.org/rfc/rfc1122.txt

[3] Cisco Networking Academy: *CCNA Exploration Course Booklet*. Course Booklets, Cisco Press, September 2009, ISBN 978-1-58713-234-8.

[4] Degermark, M.; Brodnik, A.; Carlsson, S.; aj.: Small forwarding tables for fast routing lookups. In *ACM Sigcomm*, 1997, s. 3–14.

[5] Eatherton, W.; Varghese, G.; Dittia, Z.: Tree bitmap: hardware/software ip lookups with incremental updates. 2004.
URL http://www.cs.cornell.edu/courses/cs419/2005sp/tree-bitmap.pdf

[6] Gupta, P.; McKeown, N.: Algorithms for Packet Classification.
URL
http://yuba.stanford.edu/~nickm/papers/classification_tutorial_01.pdf

[7] Gupta, P.; McKeown, N.: Packet Classification on Multiple Fields.
URL http://yuba.stanford.edu/~nickm/papers/Sigcomm99.pdf

[8] Gupta, P.; McKeown, N.: Packet Classification using Hierarchical Intelligent Cuttings. 1999.
URL http://yuba.stanford.edu/~nickm/papers/HOTI_99.pdf

[9] Kekely, M.: *Síťová vrstva TCP/IP pro FPGA*. bakalářská práce, FIT VUT v Brně, 2014.

[10] Kirsch, A.; Mitzenmacher, M.; Baohua, Y.; aj.: Using a Queue to De-amortize Cuckoo Hashing in Hardware. 2007.
URL http://www.eecs.harvard.edu/~michaelm/postscripts/aller2007.pdf

[11] Kolektív autorov: Software-Defined Networking: The New Norm for Networks. Apríl 2012.
URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[12] Kolektív autorov: The P4 Language Specification. Marec 2015.
URL http://p4.org/wp-content/uploads/2015/04/p4-latest.pdf

[13] Lampson, B.; Srinivanas, V.; Varghese, G.: IP Lookups using Multiway and Multicolumn Search. *ACM Transactions on Netrowking*, ročník 7, č. 3, 1999: s. 324–334.

[14] Nilsson, S.; Karlsson, G.: Fast address lookup for internet routers. In *IEEE Broadband Communications*, 1998, s. 11–22.

[15] Pagh, R.; Rodler, F. F.: Cuckoo Hashing. In *Algorithms - ESA 2001*, *Lecture Notes in Computer Science*, ročník 2161, Springer Berlin Heidelberg, 2001, ISBN 978-3-540-44676-7, s. 121–133.

[16] Postel, J.: User Datagram Protocol. RFC 768, 1980.
URL https://tools.ietf.org/html/rfc768

[17] Postel, J.: INTERNET PROTOCOL: PROTOCOL SPECIFICATION. RFC 791, September 1981.
URL http://www.ietf.org/rfc/rfc791.txt

[18] Pus, V.; Tobola, J.; Kosar, V.; aj.: Netbench: Framework for Evaluation of Packet Processing Algorithms. *Symposium On Architecture For Networking And Communications Systems*, 2011: s. 95–96.

[19] Singh, S.; Baboescu, F.; Varghese, G.; aj.: Packet Classification Using Multidimensional Cutting. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, New York, NY, USA: ACM, 2003, ISBN 1-58113-735-4, s. 213–224.
URL http://doi.acm.org/10.1145/863955.863980

[20] Srinivasan, V.; Varghese, G.: Faster ip lookups using controlled prefix expansion. In *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, New York, NY, USA: ACM, 1998, s. 1–10.

[21] Taylor, D. E.; Turner, J. S.: Scalable Packet Classification using Distributed Crossproducting of Field Labels. In *IEEE INFOCOM 2005*, 2005.
URL http://www.arl.wustl.edu/~jst/pubs/2005/infocom05dcfl.pdf

[22] Tobola, J.; Kořenek, J.: Effective Hash-based IPv6 Longest Prefix Match. In *IEEE Design and Diagnostics of Electronic Circuits and Systems DDECS'2011*, IEEE Computer Society, 2011, ISBN 978-1-4244-9753-9, s. 325–328.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9602

[23] Waldvogel, M.; Varghese, G.; Turner, J.; aj.: Scalable high speed ip routing lookups. In *Proceedings of the ACM SIGCOMM 97 conference on Applications, technologies, architectures, and protocols for computer communication*, New York, NY, USA: ACM, 1997, s. 25–36.

[24] Xilinx: AXI Reference Guide. UG761, 2011.
URL http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf

[25] Xilinx: LogiCORE IP AXI4-Lite IPIF (v1.01.a). DS 765, 2012.
URL http://www.xilinx.com/support/documentation/ip_documentation/axi_lite_ipif_ds765.pdf

[26] Xilinx: 7 Series FPGAs Overview. DS 180, 2015.
URL http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

[27] Yaxuan, Q.; Lianghong, X.; Baohua, Y.; aj.: Packet Classification Algorithm: From Theory to Practice. In *IEEE INFOCOM 2009 proceedings*, IEEE Communications Society, 2009.
URL http://users.ece.cmu.edu/~lianghon/docs/infocom09-hypersplit.pdf

Prílohy

Zoznam príloh

Príloha A

Obsah CD

Priložené CD obsahuje:

- experimenty/ - množiny pravidiel zapísané vo formáte vhodnom pre nástroj NetBench a výsledky experimentov.
- HW/ - hardvérová implementácia jednotky popísanej v tejto práci, spolu so súbormi potrebnými na simuláciu a testovanie.
- mapovanie/ - implementácia mapovania z jazyka P4 na správne nastavenie parametrov hardvérovej jednotky.
- tex/ - zdrojové súbory, grafy a obrázky na preklad pomocou LaTeX.
- DP_xkekel01.pdf - text diplomovej práce.
- README.txt - podrobnejší popis jednotlivých priečinkov a návody na inštaláciu potrebných knižníc, na preklad jednotlivých častí a na spustenie simulácií a testov.

Príloha B

Podrobné výsledky

Množina pravidiel	Počet unikátnych kombinácií									
	zIP+cIP	zIP+zp	zIP+cp	zIP+p	cIP+zp	cIP+cp	cIP+p	zp+cp	zp+p	cp+p
fw1_05_05	329	230	298	138	193	443	123	85	13	43
fw1_05_m05	363	264	340	157	248	574	171	104	13	43
fw1_m05_05	357	221	327	133	213	445	146	82	13	43
fw1_m05_m05	326	223	320	122	235	510	143	105	13	41
fw2_05_05	908	802	0	735	303	0	274	0	9	0
fw3_05_05_500	295	156	293	103	103	257	52	51	9	39
fw4_05_05_500	288	179	279	54	225	340	82	167	27	46
fw5_05_05_500	319	172	278	105	155	282	84	58	11	32
acl1	426	0	466	97	0	393	205	0	0	140
acl1_100	85	0	82	34	0	85	62	0	0	39
acl1_1K	493	0	619	103	0	766	297	0	0	99
fw1	128	87	112	57	90	177	66	55	13	43
fw1_100	37	23	39	12	42	68	25	37	11	26
fw1_1K	317	208	219	124	245	529	175	95	13	42
ipc1	941	216	530	152	282	228	128	84	33	53
ipc1_100	93	73	76	63	70	79	63	27	14	14
ipc1_1K	843	455	592	336	546	645	436	75	27	44

Tabuľka B.1: Počet unikátnych kombinácií hôdnot dvoch dimenzií.

Množina pravidiel	Počet unikátnych kombinácií									
	1+2+3	1+2+4	1+2+5	1+3+4	1+3+5	1+4+5	2+3+4	2+3+5	2+4+5	3+4+5
fw1_05_05	433	679	329	369	230	298	495	193	443	85
fw1_05_m05	468	783	363	432	264	340	634	248	574	104
fw1_m05_05	446	662	357	394	221	327	489	213	445	82
fw1_m05_m05	440	702	326	417	223	320	586	235	510	105
fw2_05_05	941	0	908	0	802	0	0	303	0	0
fw3_05_05_500	312	448	295	322	156	293	288	103	257	51
fw4_05_05_500	385	459	288	359	179	279	406	225	340	167
fw5_05_05_500	354	443	319	320	172	278	324	155	282	58
acl1	0	740	426	0	0	466	0	0	393	0
acl1_100	0	97	85	0	0	82	0	0	85	0
acl1_1K	0	889	493	0	0	619	0	0	766	0
fw1	161	240	128	132	87	112	188	90	177	55
fw1_100	52	80	37	49	23	39	78	42	68	37
fw1_1K	417	670	317	297	208	219	569	245	529	95
ipc1	1088	1377	941	583	216	530	371	282	228	84
ipc1_100	95	99	93	84	73	76	86	70	79	27
ipc1_1K	866	918	843	674	455	592	722	546	645	75

Tabuľka B.2: Počet unikátnych kombinácií hôdnot troch dimenzií (1=zdrojová IP adresa, 2=cieľová IP adresa, 3=zdrojový port, 4=cieľový port, 5=protokol).

Množina pravidiel	Počet unikátnych kombinácií				
	1+2+3+4	1+2+3+5	1+2+4+5	1+3+4+5	2+3+4+5
fw1_05_05	733	433	679	369	495
fw1_05_m05	830	468	783	432	634
fw1_m05_05	717	446	662	394	489
fw1_m05_m05	773	440	702	417	586
fw2_05_05	0	941	0	0	0
fw3_05_05_500	451	312	448	322	288
fw4_05_05_500	477	385	459	359	406
fw5_05_05_500	459	354	443	320	324
acl1	0	0	740	0	0
acl1_100	0	0	97	0	0
acl1_1K	0	0	889	0	0
fw1	259	161	240	132	188
fw1_100	87	52	80	49	78
fw1_1K	726	417	670	297	569
ipc1	1485	1088	1377	583	371
ipc1_100	100	95	99	84	86
ipc1_1K	927	866	918	674	722

Tabuľka B.3: Počet unikátnych kombinácií hôdnot štyroch dimenzií (1=zdrojová IP adresa, 2=cieľová IP adresa, 3=zdrojový port, 4=cieľový port, 5=protokol).

Množina pravidiel	Maximálny počet výsledkov									
	zIP+cIP	zIP+zp	zIP+cp	zIP+p	cIP+zp	cIP+cp	cIP+p	zp+cp	zp+p	cp+p
fw1_05_05	8	10	9	4	10	10	4	5	3	3
fw1_05_m05	8	9	11	4	7	10	5	5	3	3
fw1_m05_05	9	10	10	4	9	9	4	6	3	3
fw1_m05_m05	11	9	10	4	9	11	5	5	3	3
fw2_05_05	6	4	0	3	6	0	3	0	2	0
fw3_05_05_500	9	9	10	4	7	7	3	4	3	3
fw4_05_05_500	9	10	9	3	13	11	4	9	4	3
fw5_05_05_500	13	10	11	5	10	9	4	5	3	3
acl1	5	0	44	4	0	37	4	0	0	54
acl1_100	5	0	5	4	0	7	5	0	0	3
acl1_1K	7	0	7	4	0	7	4	0	0	5
fw1	6	6	6	4	7	6	4	4	3	3
fw1_100	6	7	6	4	6	4	2	4	3	3
fw1_1K	8	8	8	4	8	8	4	5	3	3
ipc1	10	7	10	4	6	7	5	6	3	5
ipc1_100	7	6	6	4	6	6	5	3	3	3
ipc1_1K	8	7	9	4	8	8	5	6	3	5

Tabuľka B.4: Maximálny počet kombinácií dvoch dimenzií, ktorým môže vyhovať jeden paket.

Množina pravidiel	Priemerný počet výsledkov									
	zIP+cIP	zIP+zp	zIP+cp	zIP+p	cIP+zp	cIP+cp	cIP+p	zp+cp	zp+p	cp+p
fw1_05_05	5	5	4	3	4	5	3	2	2	2
fw1_05_m05	4	5	4	3	4	5	3	2	2	2
fw1_m05_05	5	5	4	3	4	5	3	2	2	2
fw1_m05_m05	5	5	5	3	5	5	4	2	2	2
fw2_05_05	4	3	0	3	3	0	2	0	1	0
fw3_05_05_500	6	4	6	3	4	4	3	2	2	2
fw4_05_05_500	6	6	6	3	6	7	3	4	3	2
fw5_05_05_500	7	5	6	5	5	5	4	2	2	2
acl1	4	0	8	3	0	4	2	0	0	46
acl1_100	2	0	3	3	0	3	2	0	0	2
acl1_1K	3	0	4	3	0	4	3	0	0	3
fw1	2	3	3	2	3	2	2	2	2	2
fw1_100	3	3	2	3	2	2	1	2	2	2
fw1_1K	3	4	3	3	2	3	2	2	2	2
ipc1	4	2	3	2	3	3	3	2	2	3
ipc1_100	2	2	2	2	2	2	2	2	2	2
ipc1_1K	3	4	4	3	3	4	3	3	2	3

Tabuľka B.5: Priemerný počet kombinácií dvoch dimenzií, ktorým vyhovuje jeden paket.

Množina pravidiel	Maximálny počet výsledkov									
	1+2+3	1+2+4	1+2+5	1+3+4	1+3+5	1+4+5	2+3+4	2+3+5	2+4+5	3+4+5
fw1_05_05	14	14	8	10	10	9	11	10	10	5
fw1_05_m05	12	11	8	10	9	11	11	7	10	5
fw1_m05_05	12	12	9	9	10	10	10	9	9	6
fw1_m05_m05	13	12	11	11	9	10	11	9	11	5
fw2_05_05	6	0	6	0	4	0	0	6	0	0
fw3_05_05_500	11	9	9	9	9	10	7	7	7	4
fw4_05_05_500	16	14	9	13	10	9	16	13	11	9
fw5_05_05_500	15	13	13	10	10	11	12	10	9	5
acl1	0	38	5	0	0	44	0	0	37	0
acl1_100	0	6	5	0	0	5	0	0	7	0
acl1_1K	0	8	7	0	0	7	0	0	7	0
fw1	7	7	6	7	6	6	5	7	6	4
fw1_100	9	7	6	7	7	6	4	6	4	4
fw1_1K	11	10	8	8	8	8	7	8	8	5
ipc1	12	17	10	10	7	10	6	6	7	6
ipc1_100	7	7	7	5	6	6	6	6	6	3
ipc1_1K	9	10	8	9	7	9	8	8	8	6

Tabuľka B.6: Maximálny počet kombinácií troch dimenzií, ktorým môže vyhovať jeden paket (1=zdrojová IP adresa, 2=cieľová IP adresa, 3=zdrojový port, 4=cieľový port, 5=protokol).

Množina pravidiel	Priemerný počet výsledkov									
	1+2+3	1+2+4	1+2+5	1+3+4	1+3+5	1+4+5	2+3+4	2+3+5	2+4+5	3+4+5
fw1_05_05	6	6	5	4	5	4	4	4	5	2
fw1_05_m05	5	5	4	4	5	4	5	4	5	2
fw1_m05_05	5	4	5	3	5	4	4	4	5	2
fw1_m05_m05	6	5	5	4	5	5	5	5	5	2
fw2_05_05	4	0	4	0	3	0	0	3	0	0
fw3_05_05_500	6	5	6	5	4	6	3	4	4	2
fw4_05_05_500	8	7	6	5	6	6	7	6	7	4
fw5_05_05_500	7	6	7	5	5	6	5	5	5	2
acl1	0	5	4	0	0	8	0	0	4	0
acl1_100	0	2	2	0	0	3	0	0	3	0
acl1_1K	0	4	3	0	0	4	0	0	4	0
fw1	3	2	2	3	3	3	2	3	2	2
fw1_100	3	2	3	3	3	2	2	2	2	2
fw1_1K	3	3	3	4	4	3	3	2	3	2
ipc1	4	4	4	3	2	3	3	3	3	2
ipc1_100	2	2	2	2	2	2	2	2	2	2
ipc1_1K	3	3	3	4	4	4	4	3	4	3

Tabuľka B.7: Priemerný počet kombinácií troch dimenzií, ktorým vyhovuje jeden paket (1=zdrojová IP adresa, 2=cieľová IP adresa, 3=zdrojový port, 4=cieľový port, 5=protokol).

Množina pravidiel	Maximálny počet výsledkov				
	1+2+3+4	1+2+3+5	1+2+4+5	1+3+4+5	2+3+4+5
fw1_05_0	10	14	14	10	11
fw1_05_m05	10	12	11	10	11
fw1_m05_05	9	12	12	9	10
fw1_m05_m05	9	13	12	11	11
fw2_05_05	0	6	0	0	0
fw3_05_05_500	9	11	9	9	7
fw4_05_05_500	11	16	14	13	16
fw5_05_05_500	12	15	13	10	12
acl1	0	0	38	0	0
acl1_100	0	0	6	0	0
acl1_1K	0	0	8	0	0
fw1	5	7	7	7	5
fw1_100	7	9	7	7	4
fw1_1K	8	11	10	8	7
ipc1	14	12	17	10	6
ipc1_100	7	7	7	5	6
ipc1_1K	10	9	10	9	8

Tabuľka B.8: Maximálny počet kombinácií štyroch dimenzií, ktorým môže vyhovať jeden paket (1=zdrojová IP adresa, 2=cieľová IP adresa, 3=zdrojový port, 4=cieľový port, 5=protokol).

Množina pravidiel	Priemerný počet výsledkov				
	1+2+3+4	1+2+3+5	1+2+4+5	1+3+4+5	2+3+4+5
fw1_05_05	5	6	6	4	4
fw1_05_m05	5	5	5	4	5
fw1_m05_05	3	5	4	3	4
fw1_m05_m05	4	6	5	4	5
fw2_05_05	0	4	0	0	0
fw3_05_05_500	4	6	5	5	3
fw4_05_05_500	5	8	7	5	7
fw5_05_05_500	6	7	6	5	5
acl1	0	0	5	0	0
acl1_100	0	0	2	0	0
acl1_1K	0	0	4	0	0
fw1	2	3	2	3	2
fw1_100	2	3	2	3	2
fw1_1K	3	3	3	4	3
ipc1	4	4	4	3	3
ipc1_100	2	2	2	2	2
ipc1_1K	3	3	3	4	4

Tabuľka B.9: Priemerný počet kombinácií štyroch dimenzií, ktorým vyhovuje jeden paket (1=zdrojová IP adresa, 2=cieľová IP adresa, 3=zdrojový port, 4=cieľový port, 5=protokol).

Množina pravidiel	Najhorší prípad		
	Cyklov na paket	Priepustnosť	
		Mpaketov/s	Gb/s
fw1_m05_05	42	2.38	1.22
fw1_m05_m05	40	2.5	1.28
fw2_05_05	11	9.09	4.65
fw3_05_05_500	31	3.23	1.65
fw4_05_05_500	33	3.03	1.55
fw5_05_05_500	33	3.03	1.55
acl1	15	6.67	3.41
acl1_100	13	7.69	3.94
acl1_1K	16	6.25	3.2
fw1_100	34	2.94	1.51
ipc1_1K	22	4.55	2.32

Tabuľka B.10: Najhoršie hodnoty priepustnosti pre jednotlivé množiny pravidiel pre algoritmus HiCuts a pre frekvenciu hodín 100MHz a veľkosť paketov 64B.

Množina pravidiel	Najhorší prípad		
	Cyklov na paket	Priepustnosť	
		Mpaketov/s	Gb/s
acl1	30	3.33	1.71
acl1_100	45	2.22	1.14
fw1	30	3.33	1.71
fw1_100	33	3.03	1.55
ipc1_100	17	5.88	3.01

Tabuľka B.11: Najhoršie hodnoty priepustnosti pre jednotlivé množiny pravidiel pre algoritmus HyperCuts a pre frekvenciu hodín 100MHz a veľkosť paketov 64B.